University of Southern Queensland

Faculty of Health, Engineering and Sciences

Single Spatial Sensor Control of Robotic Arms

A dissertation submitted by

Damian Easterbrook

In fulfilment of the requirements of

ENG4111 and ENG4112 Research Project

Towards the degree of

Bachelor of Engineering (Honours) (Mechatronic)

Submitted October 2019

Abstract

This project explores the feasibility of using an accelerometer to measure displacements for the purpose of controlling robotic arms such as excavators as well as many other kinds of arm. The intention is to provide a means for the “average Joe” to easily control these devices without the need for a steep learning curve and to potentially allow experienced operators an easier pathway to control heavy earth moving machinery.

A methodology was developed where by a robotic platform including a robotic arm was designed implementing inverse kinematics for location and orientation control. Three control methods were implemented on this robotic platform including the traditional joystick control, a HTC Vive and an inertial measuring unit (IMU). Of particular interest to this project was the mitigation of the errors produced when implementing an IMU which increase quadratically for a given error input.

The errors were mitigated through several assumptions made about the use of the IMU. The resulting system resembled the gesturing system appearing in popular mobile device applications such as step counters on fitness applications. The use of the system felt like using a ratchet.

The final system was compared to the joysticks and the HTC Vive which demonstrated evidence that the IMU is effective as a method of control as the HTC Vive if somewhat slower. When compared to the joystick control method, the evidence suggests that the IMU is both faster and more accurate however, the experienced user remains untested to this date.

University of Southern Queensland

Faculty of Health, Engineering and Sciences

ENG4111 & ENG4112 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitles “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and any other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

[redacted]

Damian Easterbrook

Student Number: [redacted]

Acknowledgements

For their continuous support and encouragement during my time at university and for making every effort to help me retrain into this new field, I acknowledge my family especially my partner Julie and my children Ivy and Lucy.

For advice and feedback essential to the completion of this project, I acknowledge my supervisor Dr. Tobias Low.

Finally, for inspiration for my chosen topic, I acknowledge Jarrad Gleeson.

Damian Easterbrook

University of Southern Queensland

October 2019

Table of Contents

List of Figures

List of Tables

1 Introduction

This dissertation explores the design space of motion capturing sensors and their application to the field of robotic arms. The essential thesis is whether it is possible to perform an inverse kinematic control of a robotic arm using only a single sensor.

In order to answer this thesis, a number of steps were carried out. The first step was to carry out an extensive literature review of existing technologies to gain an appreciation for these technologies and their application in the real world. The second step was to develop a sound methodology for exploration of the technology. The third step was to develop the hardware and software required to undertake this task, a step which required the majority of the time and resources allocated to this project. The final step was to test the designs and compare and contrast the results

In developing the hardware and software, there were several processes that needed to be completed. These processes include design of a robotic platform, development of algorithms to control the robotic platform and process the data obtained from the sensors and finally protocols for transmission and reception of the data from the controllers to the robot.

1.1 Objectives

The objectives of this project were

1. Make a determination as to the accuracy and precision of information provided by a single sensor accelerometer for the purposes of displacement and rotation. The device investigated was an IMU6050.

2. Make a determination as to whether a single sensor can provide adequate information throughput to inform an inverse kinematic algorithm of its location and orientation data. The robotic arm was a PUMA clone of the researchers own design.

3. Compare the control method to existing industrial standards of control and to the commercial state of the art tracking systems. The traditional control method is the joystick and the state of the art as of the undertaking of the research was the HTC Vive.

4. Allow the exploration of other devices that incorporate this sensor. The primary example of this is the smart phone.

1.2 Expected Outcomes

This project has been designed to explore the feasibility of the use of a minimum number of sensors to safely and effectively determine the end point and orientation of a robotic arm as a generalised solution. To this effect, the minimum number of sensors required to potentially achieve this goal is a single sensor. Some research has been performed in this area however, this research has been limited to a minimum of two sensor systems. This research should provide useful information in regard to the accuracy, speed, usability and safety of the previously referred system to determine any improvements with regard to this technology and its implementation.

The specific expected outcomes of this project include as follows:

- The development of a control system which may be incorporated into a telepresence robotic system.
- The development of a standardised solution to the inverse kinematics of a PUMA robotic arm including the solution to any singularities.
- Detailed information as to the effectiveness of minimal input sensors and its application to the inverse kinematics of robotic arms.
- A treatise of this technology in comparison to current state of the art motion tracking systems and traditional control methods.

As a whole, this research should provide technical benefit to future researchers, engineers and the general “maker” aspiring to further their efforts into tracking system control and telepresence.

2 Literature Review

This literature review focuses on the existing technologies and techniques for state-of-the-art robotic arm control and provides justification for the methodological choices made throughout the project. The literature indicates that controllers of this type are considered to be “novel” with regard to excavator and plant machinery control and the industry will require some convincing to adopt this particular method of control. With regard to other forms of robotic arm, industries surrounding these devices would possibly be more accepting of this control method, in particular the industry surrounding telepresence for executive, industrial, commercial and domestic use.

2.1 Forms of robotic control

The control of a robotic arm is a complex issue. There are many considerations that need to be taken before effective control of this technology can happen.

2.1.1 Control systems

Of primary consideration is the motion of the segments and joints of the robotic arm. These need to be analysed before any form of effective control can be implemented. The inertial tensor of a robotic arm is constantly changing as the robotic arm changes poses and furthermore to this, the inertial tensor of the different segments of the arm will be different when compared in relation to each other even if the segments are identical due to the added complexity of the masses of the motors and the attachment of joints.

Standard control systems for the motion of robotic arms have been developed and are quite widely used in the industry. State of the art controllers could use either a PID controller or a State Space controller depending on the application. Feng et al. (2018) explored the development of a PID controller using a genetic algorithm. Gains were assigned pseudo randomly and data was gathered by experimentation. The genetic algorithm assigned cost scores to the various values of gain based on the results of the testing. Gains were selected based on the cost calculation, randomly modified and retested until the performance of the robotic arm was satisfactory. Kim and Oh (2013) provided additional research on humanoid control using the inverted pendulum model and a State Space controller which was modified to prioritise tasks performed by the robot. They had some success in keeping their humanoid robot standing while preforming several different gestures thus proving their control system had merit for use in the industry. Tafazoli et al. (2002) proposes an impedance-

based controller which tracks the steady state and force feedback of excavator type robotic arms and was able to demonstrate the accuracy of their system.

For this proposal, control systems of this kind will not be considered. This kind of control system will be assumed to be integral to the robotic arm and this assumption will simplify the design and experimentation resulting from this system. This will be done through the use of servos which already contain appropriate control systems of this type.

2.1.2 Automated system

Automatic control of robotic arms is not a new science. There are a multitude of examples of automated control of robotic arms dating back to the inception of their use in the automobile industry where by a robotic arm preprogramed with the required motions would replace humans on the automobile assembly line.

As far as earth moving systems go, Lever and Wang (1995) proposed an automated system for the control of an excavator for use in extreme environments such as the lunar surface. The intention was to allow automated mining of resources from the moon releasing humans from the equation. The results of their experiments were mixed depending on the aggregate being mined. For loose dirt containing very little large particles, the automated system was quite successful however, as the large particles became more numerous, this system began to fail.

The control system contained in this proposal is not designed for automation and as such, while research in this field is important, it is beyond the scope of this proposal.

2.1.3 Haptic feedback mechanisms

As humans, we rely on the sense of touch, among other senses, in order to gain knowledge of the physical location of objects we encounter in our environment. This feedback can be used in the somatosensory and psychomotor sulcus in our brains to coordinate our movements while navigating environments we encounter in our day to day lives. Robotic arms may or may not contain sensory equipment for touch and for controllers such as in this proposal a “sense of touch” would be beneficial. Such feedback from robotic devices such as arms is known colloquially as haptic feedback. It includes simple feedback devices such as vibrational mechanisms to force simulation devices that simulate a back force when a touch event occurs.

An example of a state-of-the-art haptic feedback mechanism was proposed by Morere et al. (2015). His haptic device, while not fully explored, was intended to be used on wheelchairs for disabled

people to provide them feedback for objects encountered in their environment. Their device used scanning lasers, such as those used in surveying, to scan the distance between the chair and the object and provide vibrational feedback to the user depending on the distance and orientation with respect to the chair. Their experiments were unconvincing, but ethical in that they did not produce a physical device opting for a virtual reality environment to test their device.

The HTC Vive controller does contain a vibrational mechanism which would adequately simulate haptic feedback and inexpensive vibrational devices do exist. Research of this nature would be helpful to this project, but it will not be included as the scope of this project is quite limited

2.2 Use of the controller

The device proposed in this project would be beneficial to the research carried out by Gleeson (2016) who examined the use of the Oculus Rift for use as a vision system for telepresence robotics. His research indicated that there was not enough evidence that the Oculus Rift provided any benefits for telepresence beyond several meters. In his experiments, 7 potentiometers were used to control the robotic arm which is a “poor mans” equivalent to a joystick control commonly found within industries. The research to be carried out in this project may find a place within his research by effectively replacing these potentiometers with a more intuitive system.

The particular controller outlined in this project proposal has appeared in one form when Kim et al. (2009) proposed a method of controlling a robotic arm using the human arm as an input to the system. The sensor system proposed in this proposal was rejected by Kim et al. (2009) on the grounds that the system would not adequately account for the acceleration of the sensor when compared to the acceleration of the robotic arm. In this proposal, this inadequacy will be addressed by means of separating the controller from the control system used, i.e. the controller outlined in this proposal will only be used to define the set point of the tool. It will then be up to the integrated control system, composed of an inverse kinematics algorithm, of the robotic arm to determine the appropriate torques necessary to achieve this set point. Kim et al. (2009) produced a final solution constructed using 3 sensors in total.

Another form of telepresence control can be found in Yoon and Manurung (2010) where by a telepresence robotic arm was controlled by a PC using cameras and joysticks. Again, this articles specification does not match the specifications of this particular proposal. Le (2016) provides yet another example of a robotic control mechanism which utilises a device similar to a PHANToM (a mechanical device for locating cursors in 3 dimensions). This device is applied to submarine control

with the PHANToM like device used to “push” the submarine around. Finally, Lam et al. (2009) provides another alternative to robotic control in their own form of PHANToM like device. The device proposed in this project does not utilise a PHANToM like device and as such remains unexplored.

In developing such a device, consideration needs to be made as to the fatigue encountered by operators. Fatigue causes mistakes to be made and mistakes are likely to impair costs. Oh et al. (2014) realised this and performed extensive studies on the fatigue encountered while using a particular controller.

2.3 Knowledge gap

Throughout the literature review process, there has been only one mention of the use of a single sensor controller when referring to robotic arm control. This mention appeared in a paper mentioned in the previous section by Kim et al. (2009) and was rejected and thus not developed any further. Further research is required in this field especially when considering the scope of robotic arm control requirements of the future as robotic arms become ever more popular to the wider, less experienced and less trained individuals.

The literature indicates that this form of controller may be unsuitable for use in robotic arm control due to the difficulties in obtaining acceleration information. With this project, it is intended to disprove this assumption through carefully made assumptions about how the controller will be used.

3 Methodology

This section outlines the considerations required to execute this project. These considerations include a treatise of the project methodology, testing environment and variables, risk assessment, timelines, consequential effects and intellectual property

3.1 Project Methodology

This project requires a methodological approach similar to one used to develop new and innovative technologies. This approach includes a detailed analysis of the problem and a detailed solution to the various subsystems that make up a designed system. Below is a list of analyses that will be required to be solved in order to provide a viable solution to the problem.

- Analysis and design of the robot
- Analysis of robotic arm for both forward and inverse kinematics
- Investigation of an IMU for determining displacement and rotation data
- Investigation of the HTC Vive controller for determining displacement and rotation
- Investigation of different representations of the end point
- Investigation of the underlying mathematics required to form a coherent solution
- Development of communication protocols between controller and robot
- Separation of computation between microcontroller in the controller and the microcontroller in the robot
- Development of COM port software and protocols for communication between the HTC Vive and a microcontroller
- Development of testing procedures
- Report writing

The break out of these considerations has been expanded in chapters to follow. Specific tasks have been listed in Appendix A and a completed timeline has been provided in Appendix D.

3.2 Testing Methodology

Evaluation of the effectiveness of the different devices under test will require a well-structured and repetitive test. This project requires testing with variable control devices and as such, the remainder of the environment must be kept as standardised as possible to minimise unwanted dependent variables which could potentially skew the results.

The testing methodology used in this project is outlined as follows:

- Setting a task for the robot to complete. The task performed by the robot will be completed by the operator using the different control devices under test.
- Setting the test arena. The arena needs to be effectively isolated from external influences to reduce potential dependent variables.
- Performing multiple tests with the different devices under test (i.e. multiple tests with the joysticks, the HTC Vive and the IMU control device).
- Record results of the following dependent variables
    - Task time
    - Task accuracy
- Repeat the previous 2 steps to investigate any changes to the variables when using the different control devices. Experiments should continue until a satisfactory amount of data has been collected.
- Compare the collected data for the purpose of drawing conclusions

3.3 Risk Assessment

In this section, the risk associated with a project of this kind will be assessed and the means of managing these risks in order to reduce the probability of their occurrence.

3.3.1 Hazard identification

The significance of a risk depends on 2 factors, the probability and the consequences of its occurrence. Using these metrics potential hazards can be identified and categorised. Hazards have

been identified and categorised as they relate to this project and have been listed below in order of their severity:

1. Electrocution
2. Burn injury
3. Physical injury from equipment
4. Strain injury
5. Battery explosion

As the maximum voltage utilised in this equipment is 9 V, the electrical components are all classes as low voltage however, the output impedance of the batteries used is very low so the batteries used do pose a significant threat due to electrocution. A full description of the hazards and the derivation of their categories can be found in Appendix C.

3.3.2 Management of risks

The management of the risks involved with this project have been outlined in Appendix C.

3.4 Timelines

This project occurred over the course of the 2019 academic year. The dates involved were from 25th February to 17th October which is the due date for this dissertation. A full schedule for this project has been given in Appendix D

3.5 Consequential Effects

The hardware and software contained in this project may be reused, replicated or expanded upon in order to further the understanding of the underlying thesis of this research project. It is essential that the author not be held personally liable for any damages or injuries as a result of any future research in regards to this thesis.

Every effort has been made to ensure a complete description of the design, analysis and implementation of the thesis of this project in order to mitigate any unintentional misuse of the information regarding this project. By providing thorough documentation within this dissertation, it

is intended to minimise the probability of any detrimental effects which could result from the use of any portion of this project.

3.6 Intellectual Property

All the hardware and software contained within this project will form a solid foundation for any person whom wishes to delve deeper into the field of robotics, robotic arm design and control and telepresence in general. Of great benefit is the standardised inverse kinematics which should seamlessly apply to any PUMA based robotic arm.

Students seeking to utilise or modify any of the research contained within this dissertation need only request attribution. The author only seeks appropriate recognition from those wishing to further this research.

4 Analysis and Design of the Robot

The design of the robot will take into consideration several factors which will govern the final layout of the robot. These factors are listed below:

1. The robot will be mobile
2. The robot will be able to support its own arm as well as a small load
3. The arm will be PUMA compatible
4. The tool will be a gripper capable of grasping the load in question

4.1 The chassis

The robot consists of a mobile chassis with a tracked locomotion system. Forward and reverse motion is achieved through turning both tracks at the same speed while turning is achieved through having the tracks turn at different speeds. The motors used to drive the tracks should be of sufficient torque to provide locomotion for the robot and consume a reasonable amount of power as the robot will be operating on battery power alone.

Allowing for 2.5 kg of robot and assuming a wheel radius of 30 mm and an acceleration of 100 mm/s^2, the torque of the motors can be estimated as shown below:

$$\begin{aligned} T &= F\,r \\ &= m\,a\,r \\ &= 2.5 \times 0.1 \times 0.03 \\ &= 0.0075\ N.m \end{aligned}$$

This is quite achievable using a couple of small DC motors.

Construction of the robot chassis may be simplified by using a robot chassis kit which can be purchased online at reputable robotics stores. One such chassis was found, purchased and constructed. The output torque of the motors exceeded 0.1 N.m which will easily propel the robot. The final acceleration of the robot can be calculated as shown below:

$$\begin{aligned} a &= \frac{T}{m\,r} \\ &= \frac{0.1}{2.5 \times 0.03} = 1.33\ m/s^2 \end{aligned}$$

This acceleration is more than adequate for the purposes of this project. Figure 1 below shows the assembled chassis:

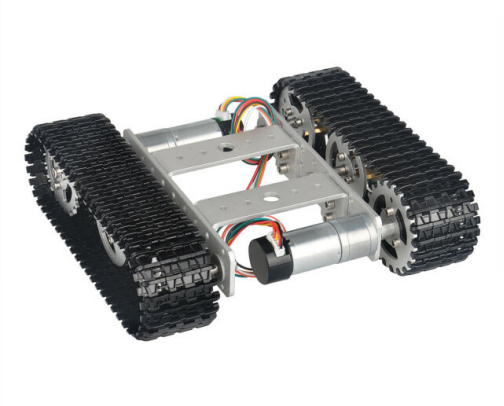


Figure 1 - Robot chassis

4.2 Robotic arm mechanical design

The completed arm is shown in Figure 2 below:

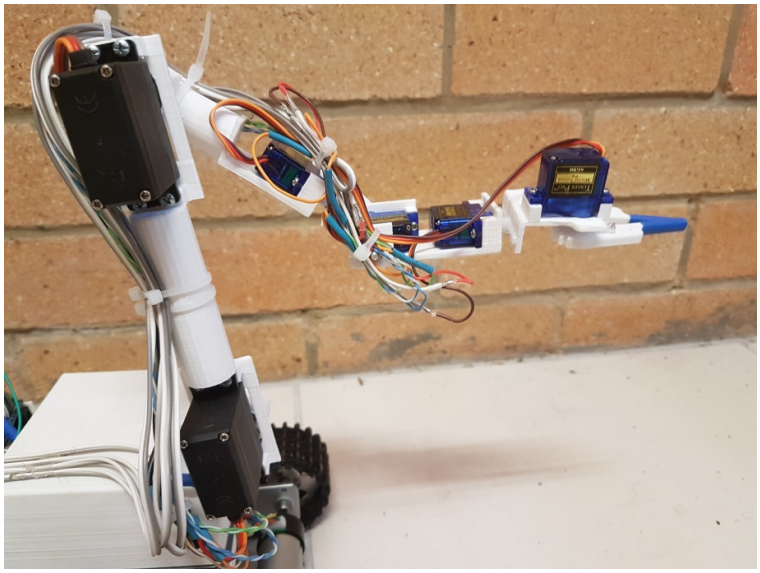


Figure 2 - Robotic arm

The arm consists of several plastic segments and is powered by a total of 7 servos. The types of servos is listed in Table 1 below:

Table 1 - List of servos

Servo	Type
1	LW-20MG
2	MG 996R
3	MG 996R
4	SG90
5	SG90
6	SG90
7	SG90

The robotic arm was constructed using a 3D printer loaded with PLA (Polylactic Acid) plastic. The density of the 3D printed material was experimentally discovered by printing a 100% infill cube 30 mm x 30 mm x 30 mm for a total volume of 27000 mm^3 or 0.000027 m^3. The cube was weighed and found to have a mass of 33 g. The density can then be calculated as shown below:

$$\rho = \frac{m}{V}$$
$$= \frac{0.033}{0.000027}$$
$$= 1222.2\ kg/m^3$$

The diameters of the circular components of the arm were 20 mm. The plastic was set to have a 1 mm wall thickness and a 20% infill density. Using these parameters and the cross section shown in Figure 3, a mass per unit length can be calculated using the following method:

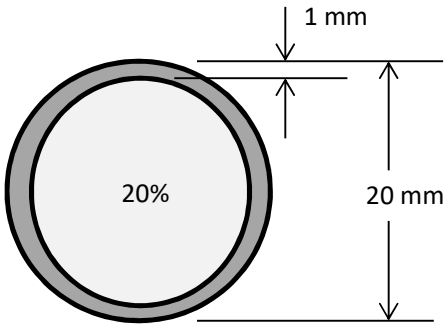


Figure 3 - Cross sectional area of an arm segment

$$A = \frac{\pi}{4}(0.02 - 0.018 \times 0.8)^2$$
$$= 0.00002463\ m^2$$

Incorporating this into the density results in:

$$Mass\ per\ unit\ length = \rho A$$
$$= 1222.2 \times 0.00002463$$
$$= 0.0301\ kg/m$$

The arm can be considered to consist of 2 segments of 150 mm length each for a total of 300 mm as an estimate. This means the moment about the arms centre of mass can be calculated as follows:

$$m = 0.3 \times 0.0301$$
$$= 0.00903\ kg$$
$$M = m\ l$$
$$= 0.00903 \times 0.15$$
$$= 0.001355\ N.m$$

Added to this is the moments created by each of the servos. The mass of each servo, its approximate distance and the moment it creates is given in Table 2 below:

Table 2 - Calculated moments of each servo

Servo	Mass	Distance	Moment
1 (LW-20MG)	86 g	0	0
2 (MG 996R)	56 g	0	0
3 (MG 996R)	56 g	150 mm	0.0084 N.m
4 (SG90)	10 g	225 mm	0.00225 N.m
5 (SG90)	10 g	250 mm	0.0025 N.m
6 (SG90)	10 g	300 mm	0.003 N.m
7 (SG90)	10 g	300 mm	0.003 N.m

The total moment about the base of the arm is the sum of all moments. This figure is 0.02076 N.m. Tower Pro (2014a) lists the torque rating of the MG 996R as 0.981 N.m which is more than adequate to drive this portion of the arm.

The next critical point of the arm is the first SG90 servo. This servo will have a negligible moment caused by the structure of the arm as well as a loading moment caused by the other 3 SG90 servos. The moment about this servo mechanism can be calculated by considering the moment of the 3 servos about the axis of this servo. This can be summarised in Table 3 below:

Table 3 - Calculated moments of servos on the first SG90

Servo	Mass	Distance	Moment
5 (SG90)	10 g	25 mm	0.00025 N.m
6 (SG90)	10 g	75 mm	0.00075 N.m
7 (SG90)	10 g	75 mm	0.00075 N.m

The total moment about the axis of this servo is the sum of each of these moments plus a negligible moment caused by the structure of the arm. This totals to 0.00175 N.m. Tower Pro (2014b) lists the torque rating of the SG90 as 0.1766 N.m which is more than adequate to drive this portion of the arm.

4.2.1 Base rotations

The base of the robot will be required to rotate in order to fulfil the specification that the robot complies with standard PUMA design specifications. To fulfil this, a Lazy Susan bearing has been used between the chassis and the first segment of the arm. This bearing is driven by an LW-20MG servo which has a torque rating of 20 kg.cm or 1.962 N.m. A picture of the base rotational components is shown in Figure 4 below:

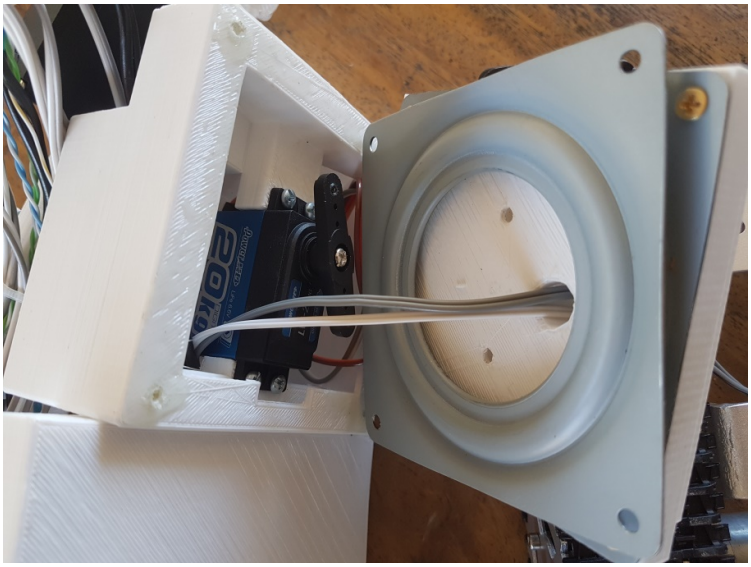


Figure 4 - Internal view of the rotational base of the robot

4.2.2 The gripper

The gripper was required to be capable of holding a load while the robot transported it over a distance. It had 5 parts in total which are listed below:

1. A driven jaw
2. A free running jaw
3. A base plate

4. A bearing for the free running jaw

5. An SG90 servo to drive the driven jaw. This servo is designated at servo 7 in Table 1 above.

The jaws are mechanically connected to each other by means of gears. As the driven jaw rotates, the gear rotates with it. This in turn rotates the free running jaw. A close up picture of the jaw mechanism is shown in Figure 5 (showing the servo and bearing) and Figure 6 (showing the gearing) below:

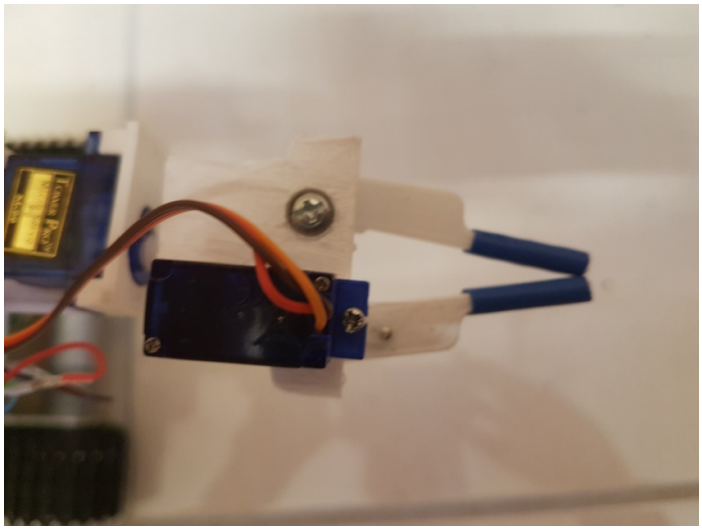


Figure 5 - Top view of the gripper

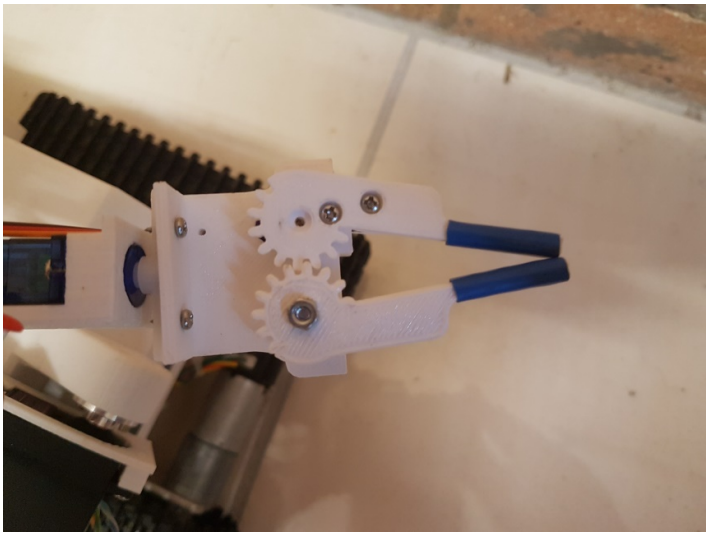


Figure 6 - Bottom view of the gripper

4.2.3 Mechanical designs

All mechanical parts of the robot with the exception of the chassis and the servos were designed in PTC Creo 4.0 and 3D printed on an Anet A8 3D printer. The engineering drawings of each part of the robot is given in Appendix E. Figure 7 below shows the assembled robot:

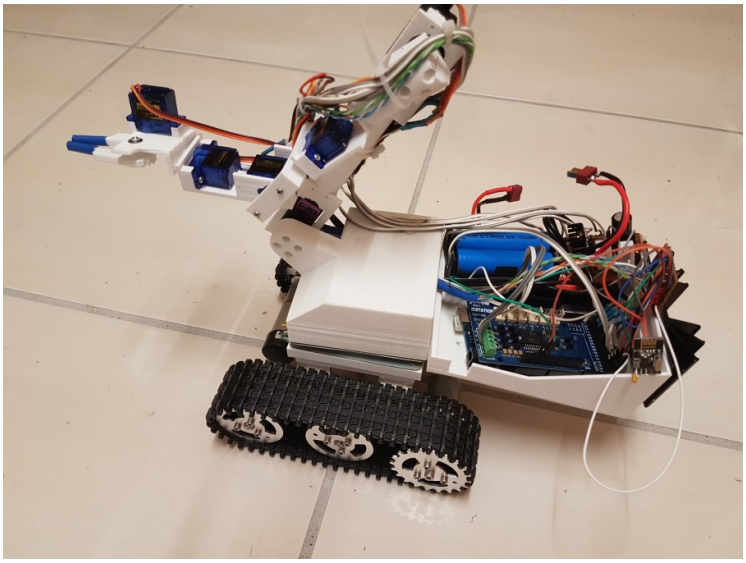


Figure 7 - Fully assembled robot

4.3 Kinematics

The calculations and equations presented in this section apply to the robotic arm utilised in this project however, the kinematic solution to this robotic arm can be adapted to any standard PUMA type robot. These equations have been derived with respect to the following mathematic models and requirements:

1. Rotation and displacement matrix mathematics
2. Quaternions
3. 3D geometric identities
4. Calculus
5. Trigonometry
6. Artificial neural networks

4.3.1 Forward kinematic calculation

There are several approaches to solve the forward kinematics of the robotic arm. This project will explore two such approaches being matrices and quaternions.

Prior to deriving the forward kinematic equations, it is important to consider the different frames of inertia applied to each of the robotic joints. For ease of calculation, it will be assumed that each of the robotic segments will align along their x-axis as shown in Figure 8 below:

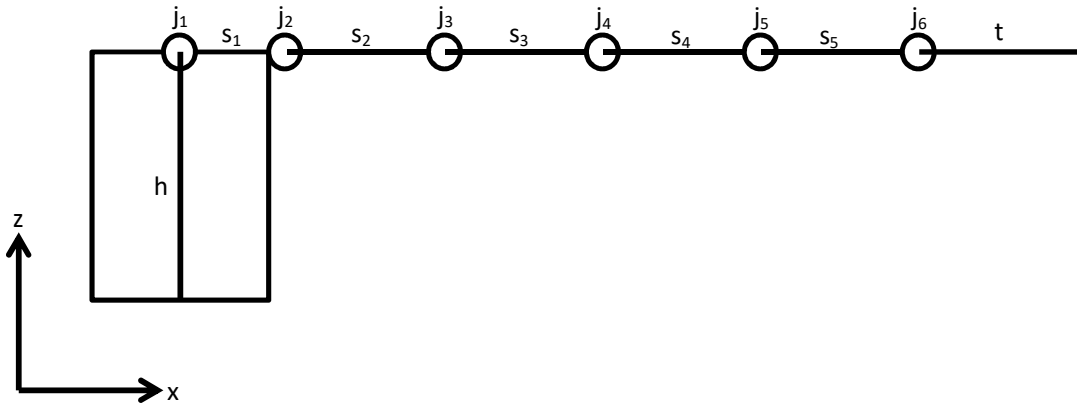


Figure 8 - Conceptual representation of the robotic arm

Table 4 below shows the axis of rotation for each of the joints.

Table 4 - Axis of rotation for each joint

Joint	Rotation Axis
1	Z
2	Y
3	Y
4	X
5	Y
6	X

Table 5 below shows the lengths of each segment in mm.

Table 5 - Lengths of each segment for the robotic arm

Segment	Length (mm)
h	132.13
s_1	54.3
s_2	150
s_3	78.5
s_4	23
s_5	58.5
t	73.5

4.3.2 Forward kinematics using matrices

Matrices are extensively used throughout the engineering fields not only for the calculation of displacements and rotations. It will be assumed that the reader will have adequate knowledge of matrix algebra and as such the background mathematics will not be included here.

The matrix transformation of the position and orientation of the end effector can be expressed as:

$$T_6^0 = T_1^0\, T_2^1\, T_3^2\, T_4^3\, T_5^4\, T_6^5 \tag{4.1}$$

Each of the transformation matrices in equation 4.1 are a combination of 2 matrices. The first matrix represents the rotation and the second matrix represents the translation along the arm segment. Thus each transformation matrix takes the form of equation 4.2 below:

$$T_{n-1}^n = R_n\ D_n \tag{4.2}$$

Where R_n is the rotation matrix for joint n and D_n is the displacement matrix for segment n.

As each of the arm segments are aligned along their x-axis, each rotation matrix is calculated relative to this frame of reference and each displacement matrix is calculated along the x-axis. Below are a series of matrices showing the rotation matrices (equations 4.3, 4.4 and 4.5) and displacement matrix (Equation 4.6) relevant to the forward kinematics:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$R_y = \begin{bmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$\mathrm{R_z} = \begin{bmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

$$D_x = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.6}$$

Where c is the cosine of the angle, s is the sine of the angle and x is the displacement.

4.3.3 Representations of Orientations

The orientation of the end effector will need representing. A naive approach would be to represent the orientation of the end effector as a simple vector however, important information will be lost if such a representation is utilised. This is demonstrated in Figure 9 and Figure 10 below

Figure 9 - Hand rotation top (image curtesy of (ClipDealer 2011))

Figure 10 - Hand rotation palm (image curtesy of (ClipDealer 2011))

As shown, a vector is able to quantify the direction the end effector is pointing to as both images would have identical direction vectors. The vector is unable to quantify the final rotation. This

implies that a simple vector is inadequate to fully quantify an orientation as this final rotation is not represented.

A second approach would be to represent the orientation through roll, pitch and yaw angles. This approach has the obvious issue of gimbal lock and as such should be used with care. In this project, roll, pitch and yaw will be used sparingly in the transmission of data over the wireless communication channel in order to save on bandwidth. For more details, see 6 Communication Protocol

As shown in Appendix H, a quaternion would be adequate to represent both the direction and rotation of the end effector. The orientation quaternion would be constructed simply by the product of all the rotations required to arrive at the final rotation.

4.3.4 Forward kinematics using Quaternions

The solution of the forward kinematics of a robotic arm can be elegantly represented using quaternions. As quaternions are not extensively used in the engineering field, some background mathematics will be required. This information is included in Appendix H.

For each joint j_n from Figure 8, a rotation quaternion q_n can be constructed from the axis of rotation and the angle of rotation. Vector representation of the different segments can be converted to quaternions of the same name.

The location of the end effector can be determined from Equation 4.7 below:

$$\vec{L} = \vec{h} + q_1\,\overrightarrow{s_1}\,q_1' + q_{12}\,\overrightarrow{s_2}\,q_{12}' + q_{13}\,\overrightarrow{s_3}\,q_{13}' + q_{14}\,\overrightarrow{s_4}\,q_{14}' + q_{15}\,\overrightarrow{s_5}\,q_{15}' + q_{16}\vec{t}\,q_{16}' \qquad (4.7)$$

Where L is the vector representation of the final location. q_1 is the rotation quaternion of joint j_1. q_{1n} is the product of the series of rotation quaternions for joints 1 through n and q_{1n}' is the conjugates of the products of quaternions for joints 1 through n. Also note the vectorisation of the lengths.

The final orientation, O, of the tool can be calculated using Equation 4.8 below:

$$O = q_6\,q_5\,q_4\,q_3\,q_2\,q_1 \qquad (4.8)$$

Where q_n is the rotation quaternion of joint j_n.

4.3.5 Inverse kinematics

There are many different ways to derive the inverse kinematics of the robotic arm, some of which may result in multiple solutions. Two such methods will be outlined here

1. Analytical solution using geometry
2. Artificial neural network

4.3.5.1 Analytical solution

There are 2 different methodologies to finding an analytical solution.

1. Simultaneous equations derived from the forward kinematics
2. Geometric analysis of the arm

Simultaneous equations rarely produce a result so the geometric analysis given an end location vector L and an end orientation quaternion O. Let L_x, L_y and L_z be the x, y and z components of the set point. Let θ_1, θ_2, θ_3, θ_4, θ_5 and θ_6 be the angles for j_1, j_2, j_3, j_4, j_5 and j_6 respectively. Starting with θ_1 a diagram can be constructed of the projection of the arm segments in x-y plane. This construction is shown in Figure 11 below:

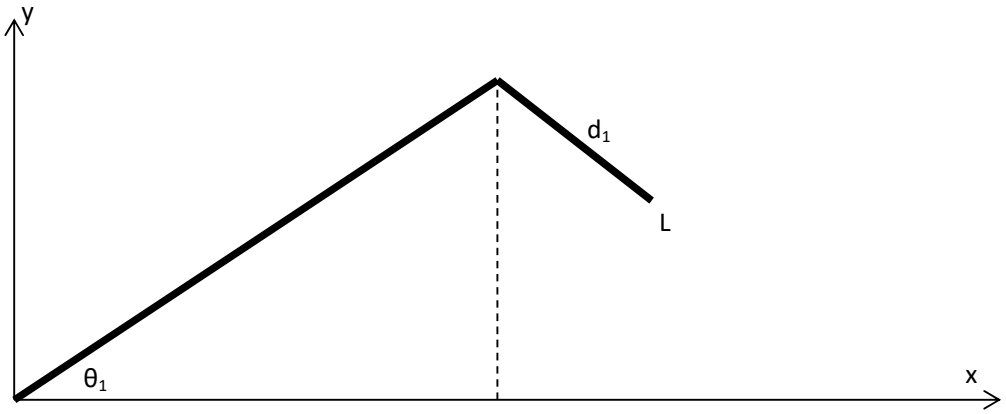


Figure 11 - Robotic arm in the x-y plane

$$d_1 = s_5 + t \tag{4.9}$$

$$\vec{V} = O\ (i)O' \tag{4.10}$$

$$\theta_1 = \tan^{-1}\left(\frac{L_y - d_1 V_y}{L_x - d_1 V_x}\right) \tag{4.11}$$

$$q_1 = \cos\frac{\theta_1}{2} + k\sin\frac{\theta_1}{2} \tag{4.12}$$

$$L' = q_1' L q_1 \tag{4.13}$$

$$V' = q_1' V q_1 \tag{4.14}$$

For θ_2 and θ_3, consider the arm in a plane defined by rotating the x-z plane through θ_1. This representation is shown in Figure 12 below:

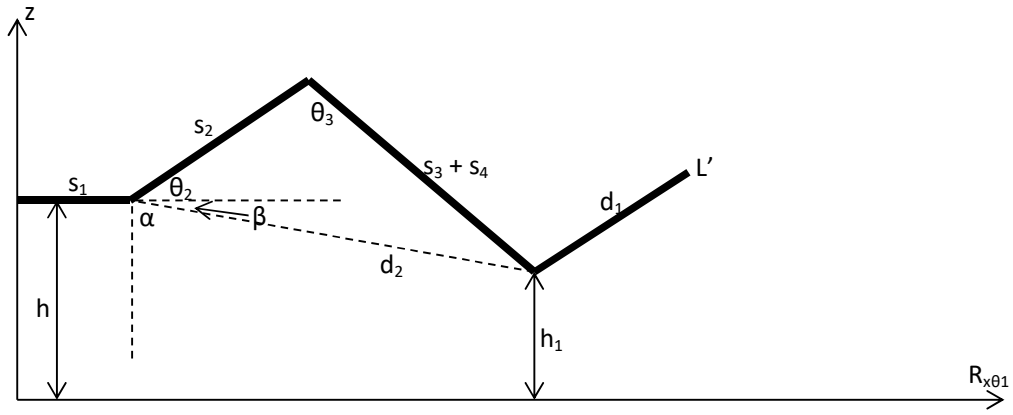


Figure 12 - Robotic arm in x-z plane rotated by ϑ_1

$$h_1 = L'_z - d_1 V'_z \tag{4.15}$$

$$d_2 = \sqrt{(h - h_1)^2 + (L'_x - s_1 - d_1 V'_x)^2} \tag{4.16}$$

$$\alpha = \cos^{-1}\frac{h - h_1}{d_2} \tag{4.17}$$

$$\theta_2 + \beta = \cos^{-1}\left(\frac{(s_3 + s_4)^2 - {s_2}^2 - {d_2}^2}{-2s_2 d_2}\right) \tag{4.18}$$

$$\theta_2 = \cos^{-1}\frac{h - h_1}{d_2} + \cos^{-1}\left(\frac{(s_3 + s_4)^2 - {s_2}^2 - {d_2}^2}{-2s_2 d_2}\right) - 90 \tag{4.19}$$

$$\theta_3 = \cos^{-1}\left(\frac{{d_2}^2 - {s_2}^2 - (s_3 + s_4)^2}{-2s_2(s_3 + s_4)}\right) \tag{4.20}$$

For angles θ_4 and θ_5, apply quaternion rotation q_{13} from Equation 4.7 to the vector $\boldsymbol{V}$ obtained from Equation 4.10 to obtain the orientation with respect to the end of segment s_4:

$$\overrightarrow{d_3} = q_{13}\, V\, {q_{13}}' \tag{4.21}$$

d_3 can be analysed in 3 dimensions to calculate θ_4 and θ_5. This is done in Figure 13 below:

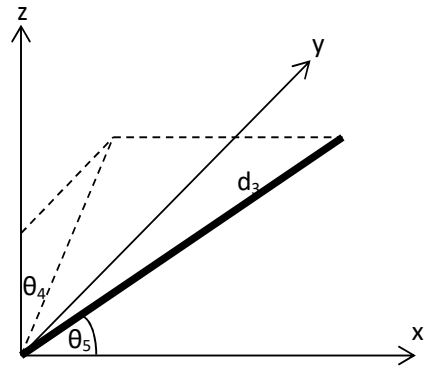


Figure 13 - Calculating ϑ_4 and ϑ_5

$$\theta_4 = \tan^{-1}\frac{d_{3y}}{d_{3z}} \tag{4.22}$$

$$\theta_5 = \cos^{-1}\frac{d_{3x}}{d_3} \tag{4.23}$$

Finally, θ_6 can be calculated from the orientation quaternion by testing the x rotational value and deriving from the w rotational value as follows in Equation 4.24

$$\theta_6 = \begin{cases} -\theta_4 & : O_x = 0 \\ 2\cos^{-1} O_w - \theta_4 & : O_x \neq 0 \end{cases} \quad (4.24)$$

The conditional test ensures that no orientation leakage occurs into θ_6 if the orientation is purely about the y or z axis (or a combination of the two).

4.3.5.2 Singularities

There are 3 possible singularities for the PUMA robotic arm. These singularities are listed below:

1. Angles θ_4 and θ_6 create a singularity when the arm is straight. There are an infinite number of solutions in this configuration.

2. Angles θ_1, θ_2 and θ_3 create a singularity when the arm is oriented front or back. There are 2 solutions in this configuration.

3. Angles θ_2 and θ_3 create a singularity when the arm is bent either above or below the set point. There are 2 solutions in this configuration

For situations 2 and 3, the resolution is a natural consequence of the limitations of the servos employed. As the servos (1, 2 and 3) are limited to angles between 0 and 180 degrees, the number of solutions is limited to 1 as the other configurations become impossible.

The solution to situation 1 comes from the design if the inverse kinematic equations which eliminates all possible configurations with the exception of 1. This is because the final orientation rotation on the x-axis is determined by the rotation of the final servo (6) while the orientation vector is a combination of the second and third to last servos (4 and 5). The effect of this is when the arm is oriented in the direction of the x-axis and there is relocation in the y-axis, the arm segments between servo 4 and 6 will rotate independent of the gripper. In this situation, the gripper is maintained by servo 6. This effect is shown in Figure 14 (arm straight) and Figure 15 (arm with y-axis displacement) below:

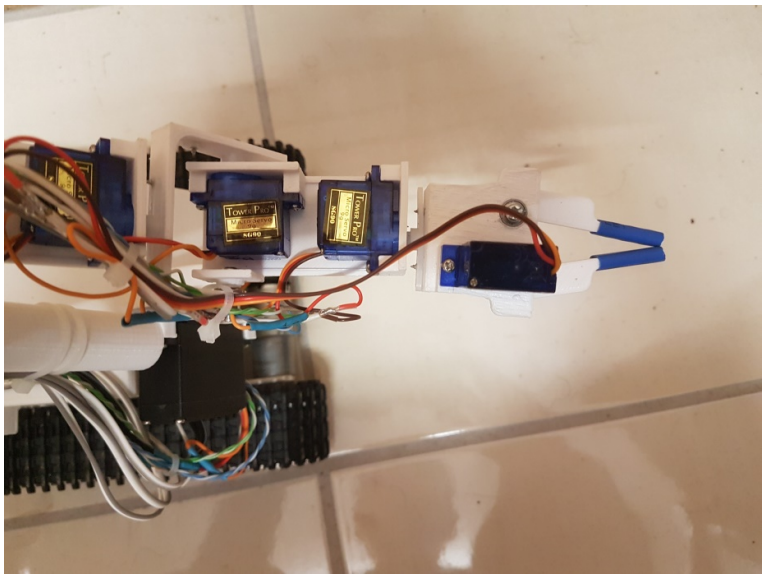


Figure 14 - Arm straight

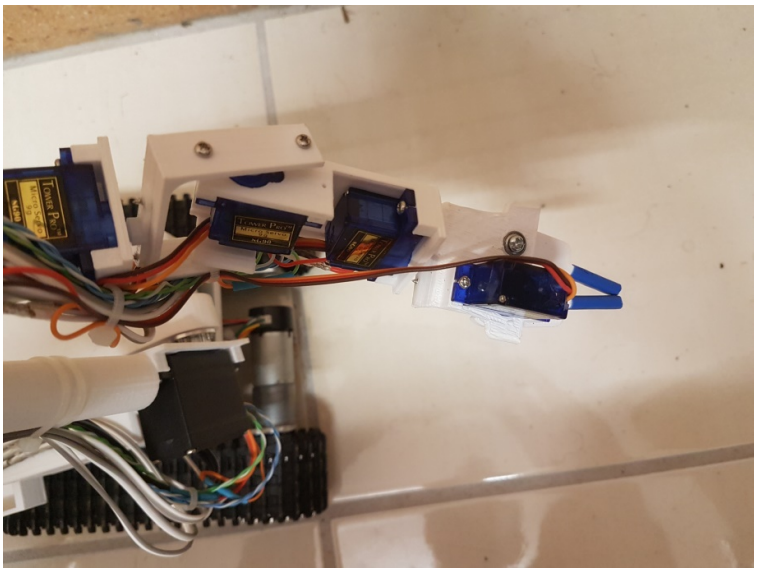


Figure 15 - Arm bent

4.3.5.3 Artificial neural network

Artificial neural networks are not often studied as part of the engineering field and as such Appendix I will give an overview of neural networks. The following section will outline how a neural network may be implemented to find a solution to the inverse kinematics problem.

A neural network N can be defined as shown in Equation 4.25:

$$N(I, L, w, \sigma) \tag{4.25}$$

Where I is the input vector, L is an ordered set of vectors representing the layers of the network, w is a set of matrices representing the weights and σ is the transfer function.

I can be defined as follows in Equation 4.26:

$$I = (x, y, z, q) \tag{4.26}$$

Where x, y and z are the coordinates of the end effector and q is a quaternion representing the orientation of the end effector.

L can be defined as follows in Equation 4.27:

$$L = \{H_1, H_2, \cdots, H_n, O\} \tag{4.27}$$

Where H_k is a vector representing a hidden layer and O is the output vector.

O can be defined as follows in Equation 4.28:

$$O = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) \tag{4.28}$$

Where θ_k is the kth joint angle.

The term σ is given as the sigmoid function shown in Equation 4.29 below:

$$\sigma = \frac{1}{(1 + e^x)} \tag{4.29}$$

The neural network requires a set of training data and validation data which can be generated from the forward kinematics equations. To prevent overfitting, the ratio of training vs validation data should be around 4:1 and the amount of training data should be just enough to produce valid results. The training data is fed into the neural network and a feed forward operation is performed. Using the results of this, an error signal is produced which is fed back through the network (as back propagation) to adjust the values of L and w. This process is repeated until all the training data has been passed into the network at which point the network is evaluated by the validation data.

4.3.5.4 Discussion of the different inverse kinematic algorithms

Both algorithms were evaluated against the following criteria:

1. Speed of execution
2. Ease of implementation
3. Ability to be run on limited processing power
4. Ability to be run in limited memory

Table 6 below shows the comparison of evaluation results against each algorithm with some comments as to their effectiveness:

Table 6 - Comparrison of different inverse kinematics algorithms

Criteria	Geometric	Neural network
1	Decent speed of execution as there are 18 trigonometry calculations, 3 square roots, 4 quaternion rotation (each consisting of 2 quaternion multiplications) as well as several general operators (+, -, *, /).	Traditionally slow. Considering there would need to be several hundred hidden neurons to provide a good enough estimation which amounts to several thousand calculations.
2	The algorithm can be directly coded by following the "recipe" provided and plugging in the values of the segments. Some math can be coded into functions for repeated code.	The majority can be coded through functions as the majority of the calculations are very similar to each other. Coefficient representation may be an issue as the majority of the coefficients will need to be stored in aligned arrays.
3	Calculations contain floating point operations which are notoriously poor at executing on integer only processors.	Calculations contain floating point operations which are notoriously poor at executing on integer only processors.
4	There are a small number of variables required for effective calculation. Should be suitable for small memory size.	Very large number of variables. Estimated size of variable storage in the kilo bytes.

Given the points above it is clear that the geometric solution is superior and should be capable of running on an Atmel ATMega2560 microcontroller as implemented on the Arduino Mega. Complete code listings can be found in Appendix G.

5 Discussion of the Controllers

In this section, the different forms of controller will be discussed. This discussion will include an analysis of the use of the IMU as a displacement measuring device as well as an analysis of the joysticks and Vive systems. Finally the function of the switches present on the joysticks and the other Vive inputs and outputs will be discussed.

5.1 Measuring Displacements with the IMU

Figure 16 below shows the final set up of the IMU measuring device. It consists of the following components:

1. A 3D printed enclosure. The engineering drawing of the enclosure is given in Appendix E.
2. A twin 18650 battery holder to provide power for the controller via 2 18650 Lithium Ion batteries.
3. 2 analogue game controller joysticks to control the robot and gripper.
4. An Arduino Nano for processing the IMU and joystick data and to provide control for the communications with the robot.
5. An nRF24l01 to communicate with the robot
6. A GY-521. This is the MPU6050 breakout board

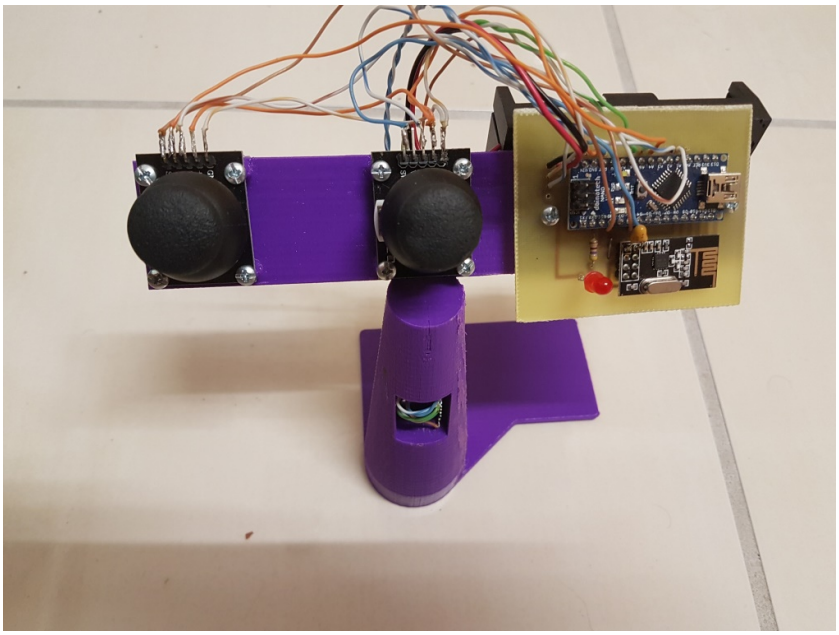


Figure 16 - Final set up of the IMU tracking device

The Inertial Measurement Unit (IMU) is a device designed to measure rotations and accelerations in space. As rotations are directly measured with the device, it becomes trivial to measure these quantities needing to only access the orientation quaternion of the device which are identical to those discussed in 4.3.3 Representations of Orientations above.

Displacement, on the other hand, will require some processing of the data received from the IMU. This processing will be discussed in the following sections.

5.1.1 Calculation of Displacement

As the data from the IMU represents the acceleration of the device, the displacement must be calculated using this acceleration data. The equation relating acceleration to displacement and time is defined as follows in Equation 5.1:

$$a = \frac{d^2x}{dt^2} \tag{5.1}$$

Solving for displacement:

$$x = \iint_{t_{k-1}}^{t_k} a\, dt^2 \tag{5.2}$$

$$= \frac{1}{2} a({t_k}^2 - {t_{k-1}}^2) \tag{5.3}$$

Equation 5.2 assumes acceleration is constant however, acceleration will not be constant and as such the above equation will not be valid for operation of an IMU. To work around this fact, a different approach will be required. For discrete data such as that obtained from the IMU, a trapezoid formula would be more appropriate.

5.1.2 Trapezoid Method

The trapezoid method initially requires the calculation of velocity and from this the calculation of displacement. Velocity can be calculated using Equation 5.4 below:

$$v_k \approx \Delta t_k \left(\frac{a_{k-1}}{2} + \frac{a_k}{2}\right) \tag{5.4}$$

Where v_k is the velocity at step k, Δt_k is the current change in time, a_{k-1} is the previous measured acceleration and a_k is the current measured acceleration.

From Equation 5.4, displacement can be calculated using Equation 5.5 below:

$$d_k \approx \Delta t_k \left(\frac{v_{k-1}}{2} + \frac{v_k}{2}\right) \tag{5.5}$$

Where d_k is the current displacement value

Substituting Equation 5.4 into Equation 5.5 we can derive displacement in terms of acceleration as shown in Equation 5.6 below:

$$d_k \approx \frac{1}{4}(\Delta t_k \Delta t_{k-1} a_{k-2} + a_{k-1}(\Delta t_k \Delta t_{k-1} + {\Delta t_k}^2) + {\Delta t_k}^2 a_k) \tag{5.6}$$

Clearly from Equation 5.6 it will be required to record historical acceleration and time measurements. For acceleration two historical measurements will be required while for time only a single historical measurement is required.

5.1.3 Error Reduction

While the above formula will give a somewhat accurate measurement of the displacement, there is an issue with this approach. Any error in the acceleration will quickly add up. An error e in the acceleration will be amplified by μe (for some $\mu > 1$) at the velocity stage which in turn will be amplified by $\mu^2 e$ at the displacement stage. Thus, errors very quickly become problematic to the measurement of displacement. Estimates have placed the time for an error to propagate to be approximately a second.

There have been several work arounds for this problem, all of them involve carefully considering the use of the displacement value. As the device proposed in this paper is to be used as a proxy locator used by a human hand, several possible optimisations become apparent.

1. The sensor need not be precise in the extreme as human hand gestures are imprecise. While this property would be desirable, it may not be possible without an expensive, military grade IMU device. With a little practice, the device has the potential to become quite precise.

2. Measuring time will need to be limited to at most 1 second to limit the error propagation in the system. As such the system will need to be operated similar to a mouse drag and time limits will be required to be enforced.

3. Accumulation of errors between gestures will need to be eliminated as such, displacements and velocities will be throttled upon completion of a gesture.

5.1.4 IMU particulars

The IMU selected for this project contains a MPU6050 integrated gyro and accelerometer shown in Figure 17 below. The particulars of this chipset are found on the datasheet (InvenSense Inc. 2013). Of particular interest is the amount of noise and the output from the digital motion processor (DMP).

Figure 17 - An example of an MPU6050 breakout board (curtesy of (Makers Electronics 2019))

The noise from the MPU6050 can be measured. Given that it will not be necessary to be highly precise with the IMU, noise can be effectively eliminated by removing the least significant bits of the accelerometer data. This reduces the range of measurement but eliminates noise entirely.

The DMP takes information from several different sensors (accelerometer, gyro and if present magnetometer) and combines them to provide a reasonably accurate quantified dataset of the state of the IMU. Obtaining data from the DMP will simplify the task of performing the double integral.

A further issue with the IMU is the time taken for it to settle. Figure 18 shows the measured accelerations from time of initialisation:

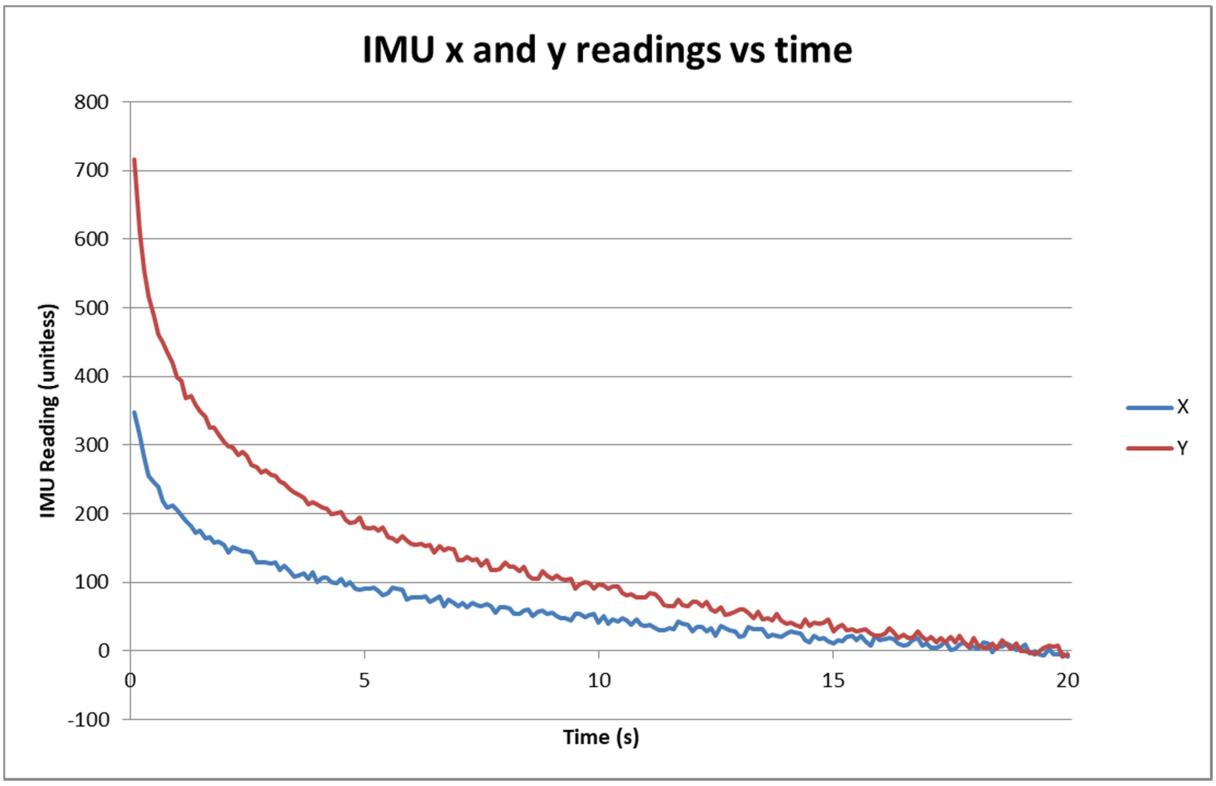


Figure 18 - IMU readings vs time

As can be seen, the IMU takes approximately 20 seconds to settle at which time the measurement of acceleration will be relative to 0.

Another particular of the MPU6050 is the z-offset setting. This setting allows the IMU to take correct measurements of the z axis (up and down) compensating for gravity. An incorrect setting here will produce large errors in the measured acceleration. This phenomenon is shown in Figure 18 above.

The final particular of the MPU6050 has to do with an issue measuring the z axis after rotation. In practice even after correct calibration of the IMU, z-axis measurements will still show an offset which will be different depending on the rotation imparted on the device. This is shown in Figure 19 below.

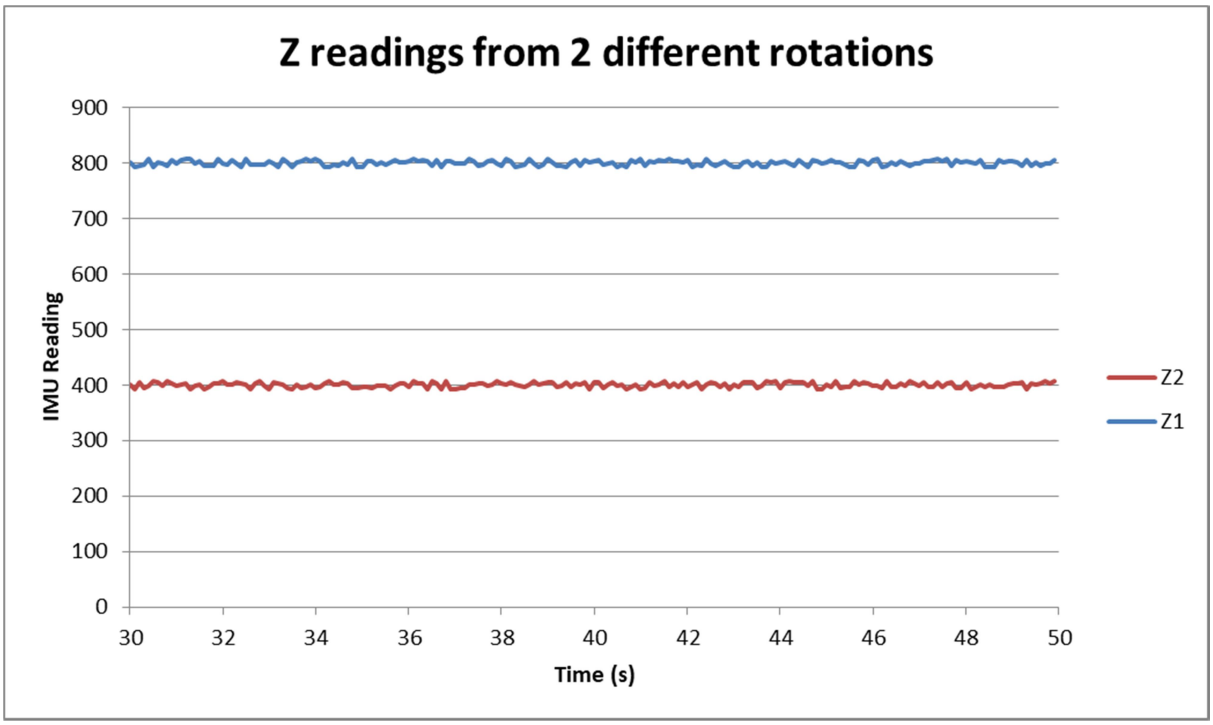


Figure 19 - Z readings from 2 different rotations

5.1.5 Displacement algorithm

Given the derivation of displacement, the assumptions and the particular properties of the IMU being used, the following algorithm can be used to determine the displacement.

```
Define storage for delta times and previous accelerations

Repeat forever

     User requested a measurement?

          Get DMP data

          Remove least significant bits

          Calculate displacement

          Check timings and complete
```

Complete code listings can be found in Appendix G.

5.1.6 Displacement scaling

Once the raw displacement value has been collected a decision needs to be made as to how much this displacement will affect the set point. That is if the IMU is moved by an amount, x, the set point will move by a different amount, y. The raw data had to be mapped to the real world. The following steps were taken to achieve this:

1. The IMU was mounted in its controller enclosure and wired up to the Arduino
2. The calculated displacement was sent through the serial of the Arduino to be displayed on the computer
3. The controller enclosure was moved along a ruler for 100 mm in each direction (x, y and z) and the raw displacement data was recorded.
4. The results were averaged to provide an estimate of the displacement per unit length.

Figure 20 below shows this set up

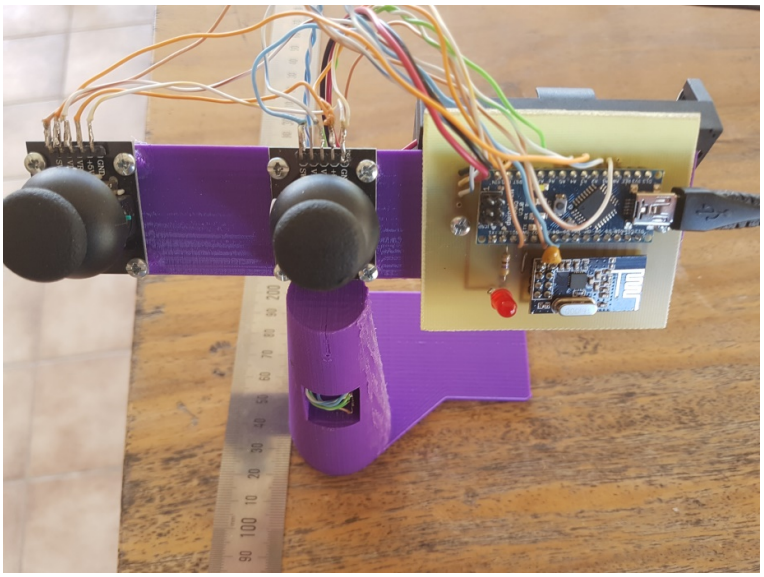


Figure 20 - IMU testing set up. IMU was run along the ruler for 100 mm and the raw data recorded

10 trials were performed for each axis and the data collected was recorded. Table 7 below shows the recorded data

Table 7 - Recorded raw data from the IMU

Trial	Raw X	Raw Y	Raw Z
1	2673	2450	2500
2	3004	2864	2908
3	2707	2704	2768
4	3025	2474	2867
5	2608	2712	2626
6	2923	2498	2936
7	2341	2852	2824
8	2457	2509	2710
9	2404	2550	3077
10	3162	2555	3221
Average	2727	2617	2844

These measurements have similar averages and it can be deduced that an average of the three averages will provide an adequate measure of the raw displacement over 100 mm. This average is calculated below:

$$s_{IMU} = \frac{2727 + 2617 + 2844}{3}$$

$$= 2729$$

This means that on average moving the controller by 100 mm yields a raw result of 2729. If the robot is to move the gripper by the same amount, a change in L can be determined by Equation 5.7 below:

$$\Delta L = \frac{R}{2.729} \tag{5.7}$$

Where R is the raw measurement from the IMU and L is in mm. In practice, this value was found to be too sensitive and was scaled back quite a lot to that found in Equation 5.8 below:

$$\Delta L = \frac{R}{350} \tag{5.8}$$

5.2 Joystick angle control

Figure 21 below shows the setup of the joystick control panel. It consists of the following parts:

1. A 3D printed platform. The engineering drawing for the platform is given in Appendix E
2. A twin 18650 battery holder to provide power for the controller via 2 18650 Lithium Ion batteries.
3. 4 analogue game joysticks, each were having 2 axes for a total of 8 axes. The control for the locomotion tracks and each servo is shared between joysticks.
4. An Arduino Nano for processing the joystick data and to provide control for the communications with the robot.
5. An nRF24l01 to communicate with the robot

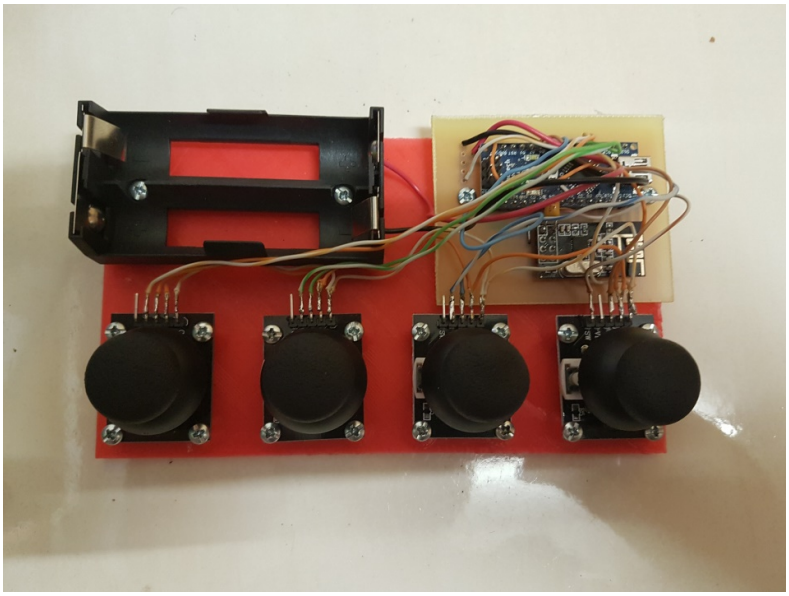


Figure 21 - Final setup of the joystick controller

Each of the joystick axes consists of a potentiometer. The potentiometer is connected between 0 V and 5 V via external pins on the joystick board. These potentials are applied to the fixed ends of the potentiometer. The wiper pin is connected to the output of the joystick via an external pin. This creates a voltage divider circuit where the output voltage falls between 0 V and 5 V depending on where the joystick is held. The joysticks are automatically centred through the use of an internal spring which makes the default output voltage approximately 2.5 V.

The outputs of the joysticks are connected to the analogue input pins of the Arduino Nano. When read by the `analogRead` statement produces a value between 0 and 1023. This value can be contained in 2 bytes, but to save on bandwidth the value is mapped to a number between 0 and 255 so it can fit within 1 byte. Figure 22 below shows the circuit diagram of a single joystick axis.

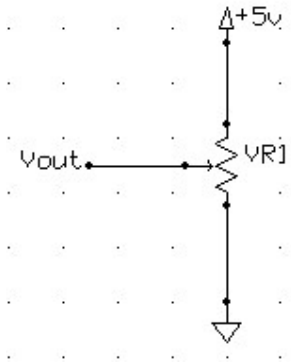


Figure 22 - Joystick circuit diagram

5.3 The HTC Vive controller

Figure 23 below shows the HTC Vive controller.

Figure 23 - HTC Vive controller (curtesy of HTC Corporation (2019))

The Vive controller senses displacements and rotations by an array of 23 sensors in total. The final position and rotation of the Vive are calculated through a complicated set of function similar to photogrammetry. The actual details of the algorithm can mostly be ignored so as to concentrate on the value provided in Unity, the graphics engine used for processing the Vive inputs in this project. The Valve Developer Community (2019) indicate that displacement input values are double precision representing the actual displacement measured in meters while orientations are double precision quaternions of the standard orientation representation as discussed in 4.3.3 Representations of Orientations above.

In order to match the values of the IMU found in 5.1.6 Displacement scaling the input from the Vive will need to be adjusted. As 100 mm displacement of the IMU caused an average of 2729 units, it follows that the Vive input will need to be adjusted by Equation 5.9 below:

$$s_{IMU} = 27290\, s_{Vive} \tag{5.9}$$

5.4 Joystick Switches and Vive functions

Each of the joysticks have a button output which has an active low signal (i.e. the output is normally high which switches to low when the button is pressed). These buttons were set to do certain critical functions depending on the controller in question. The function of the two buttons located on the IMU controller are outlined in the following list:

1. One button is required to start and finish a displacement reading. This button will set the start of the reading and while pressed will read the current displacement as the difference between the start of the reading and the current reading.

2. The other button is required to start and finish an orientation reading. Since reading the orientation affects the displacement, a separate orientation reading is required. Pressing this button reads the starting orientation quaternion and while pressed will read the current orientation as the difference between the start of the reading and the current reading.

The joystick controller only requires a single button and all other buttons are not wired into the Arduino Nano processor. The function of this button is to indicate to the robot that control has been transferred between relocating the chassis and relocating the arm.

The circuit diagram for the button is shown in Figure 24 below:

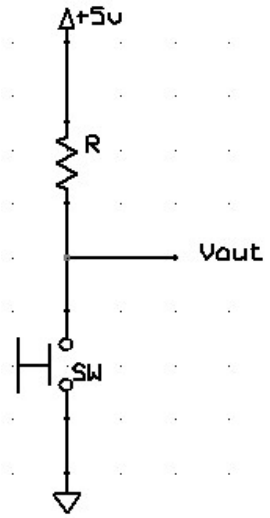


Figure 24 - Button circuit diagram

The HTC Vive controller contains several different controls. The controls and their intended function are shown in Table 8 below:

Table 8 - Vive controls and their function

Control	Function
Gripper	Start a pose measurement. When pressed Unity takes a start pose measurement and while pressed Unity reads the difference between the start pose and current pose.
Trigger	Operate the gripper on the robot.
Touch pad	Operate the locomotion of the robot chassis. It was found that this method of control was very inconsistent and was changed from track pad mode to a more consistent d-pad mode.

NOTE: The reference to *pose* in Table 8 above refers to a Unity object that contains both displacement and orientation data as a 3D vector and a quaternion.

6 Communication Protocol

This section explores the different methods of transmitting data between the controller and the robot. It then goes into selection of a suitable method. Finally the different kinds of data are broken down into a suitable packet for transmission between the controller and the robot.

6.1 Selection of Wireless Communication Hardware

The choice of wireless communication hardware is plentiful. For this project three pieces of hardware were considered. Each piece of hardware were analysed on their merit of ease of use, reliability, legality and power efficiency. Consideration was given for the fact that communication flows in one direction, from the controller to the robot. The hardware being compared for this project is listed below:

1. 433 Mhz virtual wire technology. This technology has a dedicated transmitter and a dedicated receiver. It operates on a 433 MHz carrier signal using a surface acoustic wave (SAW) as the modulation method (MuRata Manufacturing Co. Ltd. 2014).

2. nRF24l01 transceiver technology. This technology uses the same module to both transmit and receive data between two devices. It operates on a 2.4 GHz carrier and includes support for several different transceivers over separate channels (Nordic Semiconductor ASA 2006).

3. ESP8266 WiFi technology. This technology is a single chip that contains a TCP/IP stack processing capability. Commands are sent to and responses received from this chip in standard text denoted as "AT" commands. In general, the ESP8266 chip is mounted on a breakout board which includes provisions for digital control through GPIO pins and communications with a secondary microcontroller through a serial TX/RX port (Espressif Systems 2019).

6.1.1 433 MHz virtual wire

The 433 MHz virtual wire boards are shown in Figure 25 (transmitter left, receiver right) below:

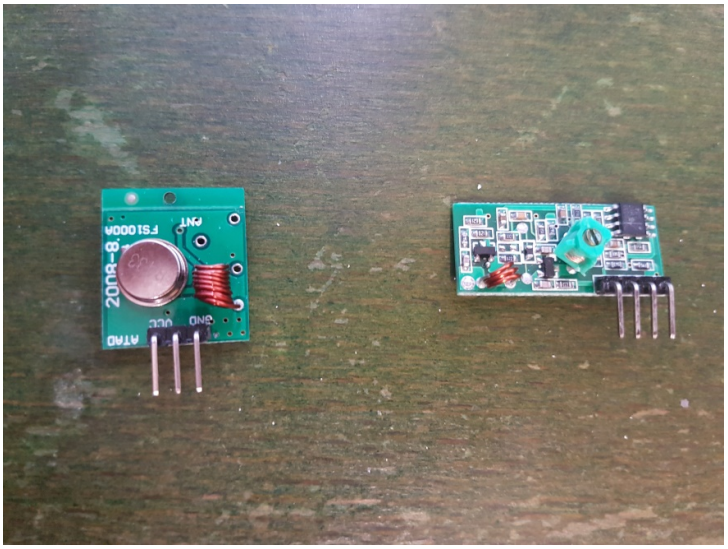


Figure 25 - 433 MHz virtual wire technology

In order to use the 433 MHz virtual wire technology, the transmitter and receiver need to be wired to the appropriate Arduino. The transmitter is wired to the Arduino on the controller while the receiver is wired to the Arduino on the robot.

6.1.1.1 Legal codes for the 433 MHz band

The Australian Communications and Media Authority (1999) are the authority that govern radio frequency transmissions in the 433 MHz band. Transmitters that use this band are classified as low interference potential devices as this RF band is shared with amateur radio (UHF) and radio location services. Accordingly any transmissions on that frequency must comply with the following criteria:

1. Limited to 25 mW output power
2. Limited to 9600 BAUD data rate to preserve bandwidth space
3. Possible limitations on the transmission times of 2 to 10 seconds
4. Must not interfere with the primary service (radiolocation)

In Australia, the 433 MHz band is not designated as Industrial, scientific and medical (ISM) critical services however, this may be the case for other countries. As such the use of the 433 MHz virtual wire technology may be impractical.

6.1.2 The nRF24l01

The nRF24l01 is shown in Figure 26 below:

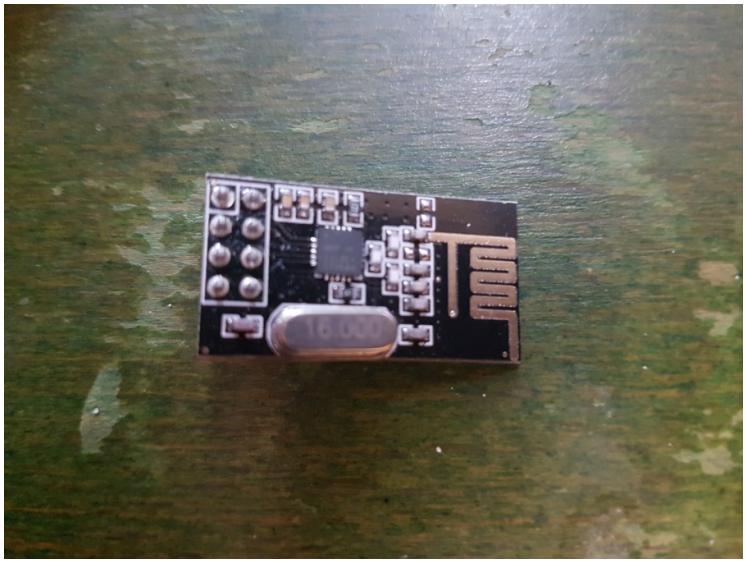


Figure 26 - nRF24l01 technology

In order to use the nRF24l01 transceiver, 2 devices will need to be wired to each Arduino. As these devices are generic in nature (i.e. both transmitter and receiver), there need only be one device for the controller and one device for the robot and it is unimportant which nRF24l01 is wired to the controller or the robot. Another advantage of this device is its capability of emulating WiFi and Bluetooth systems.

6.1.2.1 Legal codes for the 2.4 GHz band

The Australian Communications and Media Authority (2012) are the authority that govern radio frequency transmissions in the 2.4 GHz band including the operation of WiFi devices. This band is shared between WiFi devices, cordless phones, wireless medical telemetry equipment and Bluetooth. The following criteria are applicable to transmitters on the 2.4 GHz band:

1. Limited to 10 mW output power
2. Subject to Standrads Australia (2017)

3. Subject to further standards enforced through Common Law (2019) for radio communications

Despite these restrictions, the commonality and simplicity of using a 2.4 GHz band device may be advantageous.

6.1.3 The ESP8266

The ESP8266 Arduino shield is shown in Figure 27 below:

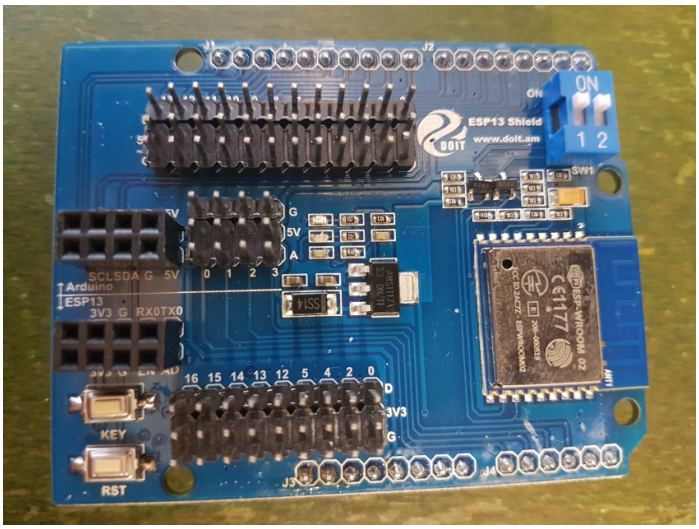


Figure 27 - ESP8266 Arduino shield

It is possible to use this device on its own for all the processing and transmission requirements of this project with the exception of analogue to digital conversion. A more straight forward approach is to use this device in conjunction with an Arduino to combine the transmission capabilities of the ESP8266 with the analogue to digital processing capability of the Arduino.

Combining this device with the Arduino requires connection of the serial ports. This is achieved by the connection of the TX of the Arduino with the RX of the ESP8266 and vice-versa. The ESP8266 will require power to operate and will need to be preconfigured to operate on the desired WiFi network. The device operates on the 2.4 GHz band and as such is also subject to the criteria outlined in 6.1.2.1 Legal codes for the 2.4 GHz band with the exception of the output power.

Commanding this device is achieved through the use of text commands transferred to the device through the serial port. The commands always begin with the text “AT” followed by a group code

and finally by a set of arguments. Received data from the device will be required to be lexically analysed and appropriately descended to determine meaning from the data.

6.1.4 Hardware selection

The selection of the hardware was based on four criteria. Each of the criteria has been discussed below:

1. Ease of use: Of the three communication devices, the 433 MHz virtual wire and the nRF24l01 are the easiest to use. This is due to the ESP8266 requirement of lexical analysis and descent to decode the output from the device. Both the 433 MHz virtual wire and the nRF24l01 have readily available libraries for immediate use.

2. Reliability: Each of these devices were fairly reliable with the ESP8266 being the most reliable followed by the nRF24l01. The least reliable device was the 433 MHz virtual wire. Research showed that the unreliability of the 433 MHz virtual wire could possibly due to the distance between the transmitter and receiver. The unreliability of the nRF24l01 is due to power limitations of the Arduino Nano. This is easily fixed using one of two methods.

    a. Construction of a 3.3 V power supply to power the nRF24l01 independently of the Arduino Nano.

    b. Installation of a decoupling capacitor across the power input of the nRF24l01 to supply power for transmission when required.

3. Legality: Both the ESP8266 and the nRF24l01 have no legal issues involved with the use of the device however, the 433 MHz virtual wire has the potential to interfere with radiolocation services which may have a legal penalty associated with any breach.

4. Efficiency: The nRF24l01 is the clear winner here drawing less than 10 mW.

Given the above analysis of each device, the obvious choice of hardware is the nRF24l01. This is the device that was used for the project after installation of decoupling capacitors in each of the devices used.

6.2 Packet Design

The nRF24l01 transmits data in the form of an array of bytes and as such for data to be transmitted from one Arduino to another requires that the data be divided into byte sized numerical

representations of the data in question. In computer science an array of bytes transmitted from one computer to another is known as a packet. The packets used in this project have been designed to ignore the overhead incurred by the hardware and driver software which simplifies the design. For the most part the packet data has been minimised with a justification provided where the data has not been justified. In all cases an identification system has been used to provide a generalised solution to packet decoding (i.e. the receiver calls a decoding function dependent on the identification of the packet)

6.2.1 Joystick packet

To identify that the incoming packet originated from a joystick array an identifier of 0x01 has been assigned. The data from the joystick and its size is summarised in Table 9 below:

Table 9 - Joystick data summary

Data	Size
Joystick 1 – y	10 bits
Joystick 1 – x	10 bits
Joystick 2 – y	10 bits
Joystick 2 – x	10 bits
Joystick 3 – y	10 bits
Joystick 3 – x	10 bits
Joystick 4 – y	10 bits
Button press	1 bit

Given the nRF24l01 transmits data as an array of bytes, it follows that each of the 10 bit values assigned to each axis of the joysticks can be reduced to 8 bits without any significant loss in performance. This has the effect of reducing the packet size to only 9 bytes as opposed to 16 bytes. Prior to construction of the packet, the microcontroller will need to truncate the 2 least significant bits of the joystick values. The arrangement of the final packet is shown in Figure 28 below:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
ID	BTN1	Y1	X1	Y2	X2	Y3	X3	Y4

Figure 28 - Joystick controller packet diagram

6.2.2 IMU and Vive packet

To identify that the incoming packet originated from the IMU or the Vive controller an identifier of 0x02 has been assigned. Additionally, the orientation of the controller was converted to roll, pitch and yaw angles in order to conserve bandwidth. The data from the IMU is shown in Table 10 below:

Table 10 - IMU data summary

Data	Size
Button 1	1 bit
Button 2	1 bit
Left track	10 bits
Right track	10 bits
Gripper control	10 bits
X displacement	32 bits
Y displacement	32 bits
Z displacement	32 bits
Roll angle	32 bits
Pitch angle	32 bits
Yaw angle	32 bits

Again, the outputs from the joysticks can be converted to 8 bit values saving on 2 bytes each. This process is identical to the procedure outlined in 6.2.1 Joystick packet above. In addition to this, each of the displacements and orientations can be converted to 16 bit values which saved a total of 12

bytes. Prior to construction of the packet, the microcontroller will need to truncate the 16 most significant bits of the displacement and orientation values. Button 1 and button 2 could have been combined into a single byte however, this was not implemented.

The data from the Vive is shown in Table 11 below:

Table 11 - Vive data summary

Data	Size
Gripper	1 bit
Left Track	32 bits
Right Track	32 bits
Trigger	32 bits
X displacement	32 bits
Y displacement	32 bits
Z displacement	32 bits
Roll angle	32 bits
Pitch angle	32 bits
Yaw angle	32 bits

There is a lot of data saving potential from the output of the Vive. As the left and right tracks and the trigger can be reduced to 8 bits each, this saves 9 bytes from the packet. Additionally, the displacements and orientations can be truncated to 16 bits each saving a further 12 bytes. If the gripper data is repeated once, the resulting packet becomes identical to the IMU packet and as such both controllers will output identical datagram configurations. The arrangement of the IMU and Vive packet is shown in Figure 29, Figure 30, Figure 31 and Figure 32 below:

Byte 1	Byte 2	Byte 3	Bytes 4 & 5	Bytes 6 to 17
ID	BTN1/GRP	BTN2/GRP	TRACK	LOC/ORI

Figure 29 - IMU and Vive packet diagram

Byte 4	Byte 5	Byte 6
L TRACK	R TRACK	GC

Figure 30 - TRACK break down

Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12
XMSB	XLSB	YMSB	YLSB	ZMSB	ZLSB

Figure 31 - LOC break down

Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18
ROLLMSB	ROLLLSB	PITMSB	PITLSB	YAWMSB	YAWLSB

Figure 32 - ORI break down

Table 12 below lists the codes used in the above diagrams along with an explanation of what the code stands for:

Table 12 - Codes used in the packet diagrams

Code	Explanation
ID	The identifier of the packet to allow the robot to know which type of controller is being used
BTNx	The state of a button
GRP	The state of the Vive gripper
Yx	The value of the y axis of a joystick
Xx	The value of the x axis of a joystick
L/R TRACK	The value to be passed to the track control for locomotion of the chassis
GC	The control signal for the arms gripper
X/Y/Z MSB	The most significant bits of the x, y or z location
X/Y/Z LSB	The least significant bits of the x, y or z location
ROLL MSB/LSB	The most or least significant bits of the roll angle
PIT MSB/LSB	The most or least significant bits of the pitch angle
YAW MSB/LSB	The most or least significant bits of the yaw angle

6.2.3 Roll, pitch and yaw from orientation quaternion

The values for roll pitch and yaw can easily be gotten from the orientation quaternion. As the quaternion represents the rotation required to achieve a desired orientation, the values for roll, pitch and yaw are intrinsically embedded within the quaternion. A method for extracting them will be required. This method will require storage of 2 quaternion variables. The first quaternion stores the reference orientation and the second stores the current orientation which gets updated continuously while a measurement is taken. To get the roll, pitch and yaw values, the current orientation needs to have the reference removed from it. Equation 6.1 below shows the formula to achieve this effect:

$$q = {q_m}^{-1} q_{ref} \tag{6.1}$$

Where q is a quaternion containing the roll, pitch and yaw values, q_{ref} is the reference quaternion and q_m is the measured quaternion.

The values of roll, pitch and yaw will not be contained in the i, j and k coefficients of q, but will be contained in different coefficients. Table 13 below shows the locations of the roll, pitch and yaw values

Table 13 - Getting roll, pitch and yaw from the quaternion

Rotation	Value in q
Roll	k coefficient
Pitch	i coefficient
Yaw	j coefficient

These values will be in half radians which is not very suitable for transmission in the proposed packet. A conversion equation is shown in Equation 6.2 which will give the angles in tenths of a degree.

$$A_{deg} = \frac{3600 A_{rad}}{\pi} \tag{6.2}$$

Where A_{deg} is the angle in tenths of a degree and A_{rad} is the angle in radians.

7 Electrical System

This section provides and explanation of the electrical system including the power requirements and design and how the different electrical components are connected together. Complete circuit diagrams are provided in Appendix F.

7.1 Power systems

The majority of the devices in this project can be considered to be portable. The exception to this is the Vive controller which, from a power supply perspective, can be considered to be a "black box" in that the means of the Vive getting power has a readily made solution by the manufacturer in the form of a USB charging system. For the other devices, a battery system was required to provide power to the devices. Table 14 below shows the batteries that were used along with an explanation of their specifications and the use of the battery in the project.

Table 14 - Battery technology used in the project

Battery	Specification	Use in project
18650 Li-ion	3.4 Ahr 3.7 V nominal	Power for Arduino Nano in controllers, the Arduino Mega in the robot and the motors in the chassis.
USB power from computer system	5 V, 500 mA max	Power for the Arduino Mega that provides radio forwarding from the Vive to the robot.
1162103 Li-ion	10 Ahr 3.7 V nominal	Originally for power to the servos. These batteries turned out to be under powered and were changed to a different battery system
LiPo RC battery	2.2 Ahr 7.4 V nominal, 30C	This was the replacement for the servo batteries. This battery is capable of supplying a theoretical 66 A.

The Arduino power system is internally contained to provide the necessary voltages required for the Atmel chip and the USB to Serial chip. This power is supplied from the batteries via a simple linear regulator. The batteries need only be connected between the Vin and the GND pins of the Arduino to allow the regulators to perform their function. The servos on the other hand require a very specific voltage range in order to function correctly as well as a stall supply current. The voltage range and supply current for each servo is shown in Table 15 below

Table 15 - Voltage and current ratings of each servo

Servo	Voltage range	Current supply
LW-20MG	4.8-6.6 V	2.7 A
MG 996R	4.8-6 V	2.5 A
SG90	4.8-6 V	0.8 A

As each servo loads, it will pull current from the power supply circuit which will affect the available voltage for the other servos. This means a specialised power supply will need to be designed and constructed to power the servos.

7.1.1 Robot power system

Figure 33 below shows a block diagram of the power supply system for the servos:

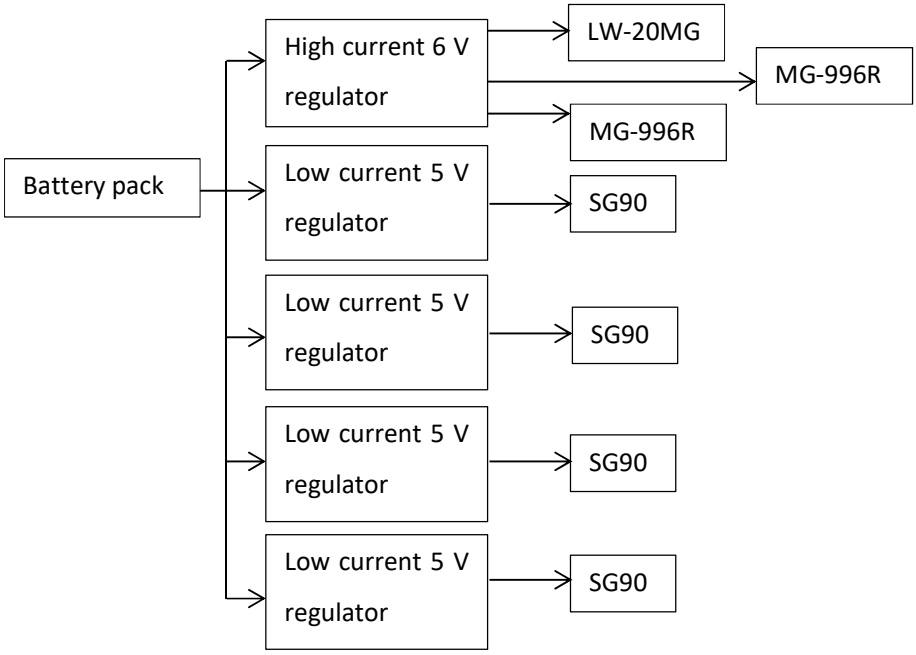


Figure 33 - Block diagram of servo power system

As discussed in the previous section, each servo will need to have a specialised power supply in order to operate correctly. As the high torque servos (LW-20MG and MG-996R) require a large amount of current, these will be supplied from the same power supply with a high enough decoupling capacitor to supply stall currents as needed. The remaining SG90 servos will be supplied 5 V each from separate regulators such that the high current requirements of any other servo will not affect the power supplied to the servo. Experimentally, this worked very well and it was found that the high torque servos (LW-20MG and MG-996R) were very tolerant to the stall currents of other servos on the same supply however, the SG90 servos were quite sensitive to changes in voltage as the current supplied to the high powered servos increased. This resulted in a "jittering" of the robot arm wherever an SG90 servo was used including the gripper. This could potentially cause inaccuracies in position, orientation as well as the loss of the load.

Power was supplied to the servos through the use of linear regulators. An LM317 adjustable regulator set to 6 V with a bypass transistor for current amplification was used to supply the high torque servos. Each SG90 servo was powered via an LM7805 regulator. The full circuit diagram is given in Appendix F.

7.2 Interconnection of Hardware

The interconnection of the hardware will need to be considered. Each device of the system will be considered separately in the sections below while a complete circuit diagram is provided in Appendix F.

7.2.1 Joystick controller connections

The joystick controller requires analogue signals for each of the joystick axes as well as a connection to the nRF24l01. These connections are done via the Arduino Nano. The joystick axes are each connected to a single analogue pin on the Arduino Nano to allow the Arduino Nano to read the position of each joystick axis in turn. The nRF24l01 is connected to the Arduino Nano via the serial peripheral interface (SPI) port. The pin requirements for this are fairly specific and are common to all the Arduino Nanos used in this project. Finally, the Arduino will need to be able to read a button press through a digital input pin. Table 16 below shows the connection of each piece of hardware to the associated pins of the Arduino Nano:

Table 16 - Joystick controller hardware connection pins

Hardware	Arduino Nano pin
Joystick 1 y	A1
Joystick 1 x	A0
Joystick 2 y	A3
Joystick 2 x	A2
Joystick 3 y	A5
Joystick 3 x	A4
Joystick 4 y	A6
Button	D3
nRF24l01 MOSI	D11
nRF24l01 MISO	D12
nRF24l01 SCK	D13
nRF24l01 CE	D9
nRF24l01 CSN	D10

7.2.2 IMU controller connections

The IMU requires 3 analogue signals for the joystick axes controlling the tracks of the robot chassis as well as the joystick axis controlling the gripper. Each axis of the joysticks is connected to an analogue pin of the Arduino Nano to allow the Arduino Nano to read the position of each joystick axis in turn. The nRF24l01 is connected to the Arduino Nano via the SPI port. The pin connections are identical to those specified in 7.2.1 Joystick controller connections above. The IMU communicates using the inter-integrated circuit (I^2C) protocol. This protocol is present on the Arduino Nano through a few pin connections which are specific and common to all Arduino Nanos used in this project. In addition to these connections, the Arduino Nano will be required to read the state of two buttons as

well as provide an output to an LED. The LED is present to indicate the status of the IMU as to whether it has settled and whether the user is taking a measurement. Table 17 below shows the connection of each piece of hardware to the associated pins of the Arduino Nano:

Table 17 - IMU controller hardware connection pins

Hardware	Arduino Nano pin
Left track	A0
Right track	A1
Gripper	A3
Button 1	D4
Button 2	D5
nRF24l01 MOSI	D11
nRF24l01 MISO	D12
nRF24l01 SCK	D13
nRF24l01 CE	D9
nRF24l01 CSN	D10
IMU SCL	A5
IMU SDA	A4
IMU INT	D2
LED	D3

7.2.3 Serial to nRF24l01 interface

The Vive controller interfaces with the computer through a Bluetooth connection which passes controller information to Steam VR which is chained through the Unity engine to process the inputs

from the Vive and output the appropriate packet to one of the USB ports. This serial signal needs to be repeated over an nRF24l01. There are a number of ways to do this however, as this project has used the Arduino extensively, it is fitting that this task be performed by the Arduino. One main issue with this is that debugging can be inhibited by Unitys use of the USB port to communicate the packet to the Arduino. The solution to this is to utilise a USB to serial module such as the one shown in Figure 34 below:

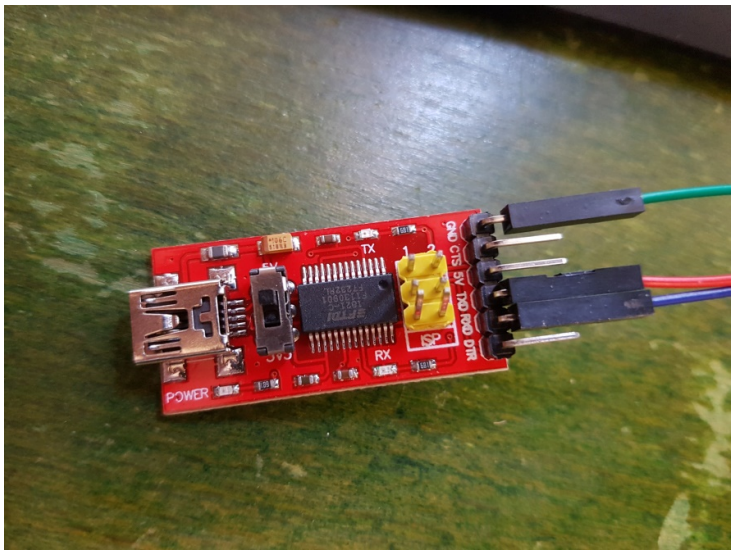


Figure 34 - USB to Serial module

The use of this module allows Unity to connect to the Arduino through it while leaving the original serial port unused and available for debugging. The USB to serial module simply connects to any available serial port via the TX and RX pins. It is powered through the USB port so only a common ground connection is required to provide a reference voltage.

In addition to the USB to serial module, the repeater requires an nRF24l01. As discussed previously the nRF24l01 connects through an SPI port however, the inclusion of the USB to serial module means that an Arduino Mega would be more suitable to use instead of an Arduino Nano. This removes the requirement of software driven serial port allowing full hardware serial capabilities for communicating with the PC. The SPI port of the Arduino Mega differs from the Arduino Nano only in pin location, not function. Table 18 below shows the connection of each piece of hardware to the associated pins of the Arduino Mega:

Table 18 - Serial to nRF24l01 hardware connection pins

Hardware	Arduino Mega pin
USB-Serial TX	D15
USB-Serial RX	D14
nRF24l01 MOSI	D50
nRF24l01 MISO	D51
nRF24l01 SCK	D52
nRF24l01 CE	D40
nRF24l01 CSN	D53

7.2.4 Robot connections

The robot requires an nRF24l01 in order to receive the communication packets. As has been stated previously, the nRF24l01 connects to an Arduino via the SPI port which is specific and common between similar Arduinos. As the Arduino that will be used for controlling the robot is an Arduino Mega, the SPI port pins will be identical to the USB to nRF24l01 interface circuit. In addition to this, the robot will require several outputs specifically outputs to each of the servos as well as outputs to the chassis track motors. A standard motor shield was used to control the motors meaning the pins that control the servos need only be designated as the pins for controlling the motors have been predesignated. Finally, provision for 3 LED indicators has been included to indicate the state of the robot which has been connected to 3 separate digital pins of the Arduino Mega. Table 19 below shows the connections of each piece of hardware to the associated pins of the Arduino Mega:

Table 19 - Robot hardware connection pins

Hardware	Arduino Mega pin
nRF24l01 MOSI	D50
nRF24l01 MISO	D51

nRF24l01 SCK	D52
nRF24l01 CE	D40
nRF24l01 CSN	D53
Servo 1	D22
Servo 2	D23
Servo 3	D24
Servo 4	D25
Servo 5	D26
Servo 6	D27
Servo 7	D28
LED 1	D36
LED 2	D37
LED 3	D38
Motor A PWM	D3
Motor A Dir	D12
Motor A Brake	D9
Motor B PWM	D11
Motor B Dir	D13
Motor B Brake	D8

8 Coherent System Design

This section describes how the overall system of the project was divided into coherent subsystems and how those subsystems interact with each other to perform the task of the project. This includes consideration for how the robot chassis moves in its environment, how the outputs from the inverse kinematic algorithm are mapped to the input values to the servos and which microcontroller performed specific tasks.

8.1 Robot Locomotion Considerations

The robot locomotion system needed to be properly designed in order to give accurate drive to the motors that move the chassis around the arena. There are a number of issues to consider when designing this system.

8.1.1 Mapping of joystick values to motor control signals

The input to the robot locomotion system consists of 2 values between 0 and 255. The first value is for the left track and the second value is for the right track. The output to the motor shield is 6 values. These values control the pulse width modulation (PWM) between 0 and 255, direction of 0 or 1 and brake of 0 or 1 for the motors. It was necessary to convert the 0 to 255 number into a combination of the 3 output values. Additionally, the analogue joysticks, when not in use would not give a value exactly centred as 127 or 128 but would give a value between 107 and 148.

An algorithm was needed to be developed to map the joystick values to the 6 motor drive values. The following algorithm was used to achieve this effect (only the algorithm for one drive is given, the other drive is identical):

```
Set the brake to 0 (no brake)

Subtract 127 from the input signal

If the input signal is greater than 40

      Set the output drive to the input signal times 2 plus 1

      Set the direction to 0 (forward)

Else, if the input signal is less than -40

      Set the output drive to the input signal times 2 plus 1 and
      negate it

      Set the direction to 1 (reverse)

Else

      Set the output drive to 0

      Set the direction to 0

      Set the brake to 1 (brake on)

Write the values to the appropriate pins
```

This algorithm gives a buffer region of 30% of full signal where there is no output drive which was sufficient to cover the error in the joystick centring system. Complete code listings can be found in Appendix G.

8.1.2 Mapping of Vive touch pad and d-pad values to joystick values

The output from the touch pad is a 2 axis double precision floating point value between -1 and 1 such that one double is assigned to the x axis and the other is assigned to the y axis. The output from the d-pad contains 4 bits, one for each major direction (up, down, left and right). Diagonal directions are possible by setting 2 of the bits at the same time. For example if the direction is up and left, both up and left bits will be set. These signals need to be converted to the equivalent left and right track joystick values of 0 to 255 each.

8.1.2.1 Touch pad

The touch pad offers a means of providing a variable track drive through the analogue nature of the output from it. Converting the touch pad output does create a challenge. Figure 35 below shows the ideal scheme for a touch pad to track drive system.

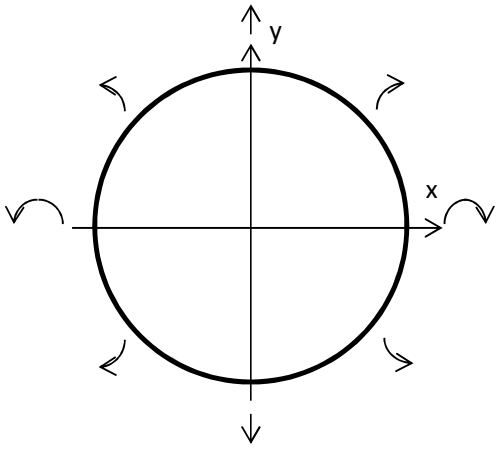


Figure 35 - Robot chassis direction according to position of touch pad

Observing the resultant directions from Figure 35, Table 20 below shows the track drive for each form of movement:

Table 20 - Touch pad to track drive conversion

Direction	Left track	Right track
Forward	Forward	Forward
Reverse	Reverse	Reverse
Full left	Reverse	Forward
Full right	Forward	Reverse
Forward left	Off	Forward
Forward right	Forward	Off
Reverse left	Off	Reverse
Reverse right	Reverse	Off

A keen eye will notice that the track drive shown in Table 20 indicates that going from forward to reverse while turning causes the opposite track drive to suddenly reverse. This situation caused an issue with testing and as a result the d-pad option was implemented and further tested. For more details, see 9 Testing Results below.

The algorithm used to convert the touch pad to left and right track drive is listed below:

```
Set both track drives to y times 127 plus 127

Subtract x times 127 from the right track drive

Add x times 127 to the left track drive

Limit both track drives to a value between 0 and 255

Negate both track drives
```

8.1.2.2 D-pad

While a variable track drive is desirable, it is impossible using the d-pad and as such, the d-pad will switch the tracks from no drive to full drive. The algorithm to convert the d-pad values to the equivalent left and right track joystick values is as follows:

```
Get the input signals from the Vive controller

If the up button is pressed

      If the left button is pressed

            Set the right drive to 0 and the left drive to 127

      Else if the right button is pressed

            Set left drive to 0 and right drive to 127

      Else

            Set both drives to 0

Else if the down button is pressed

      If the left button is pressed

            Set the left drive to 127 and the right drive to 255

      Else if the right button is pressed

            Set the left drive to 255 and the right drive to 127

      Else

            Set both drives to 255

Else if the left button is pressed

      Set the right drive to 0 and the left drive to 255

Else if the right button is pressed

      Set the right drive to 255 and the left drive to 0

Else

      Set both drives to 127
```

8.2 Servo drive

The servo drive signals are measured in microseconds. Each different servo will have different timing ranges for -90 to 90 degrees and these timing ranges are listed below in Table 21:

Table 21 - Timing values for each servo in microseconds

Servo	Minimum timing	Maximum timing
LW-20MG	450	2650
MG-996R	450	2650
SG90	700	2300
SG90 (gripper)	500	1350

The differing values for the gripper in Table 21 above is due to the gripper not requiring the full 180 degrees of travel.

The arrangement of the servos further complicates this as one angle for one particular servo will result in one particular angle on the physical robotic arm while the same angle on a different servo will result in a different angle on the physical robotic arm. This means that each individual servo requires a different mapping from their input angle through to their drive value. Each of the servo mappings are shown in Figure 36 through Figure 42 below:

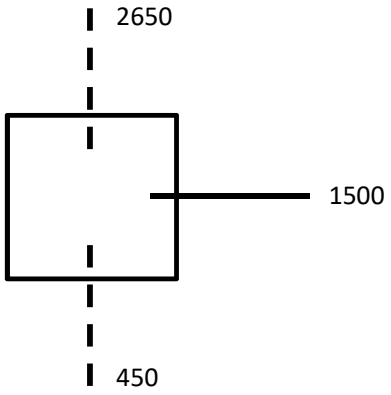


Figure 36 - Servo 1: Viewed from the top

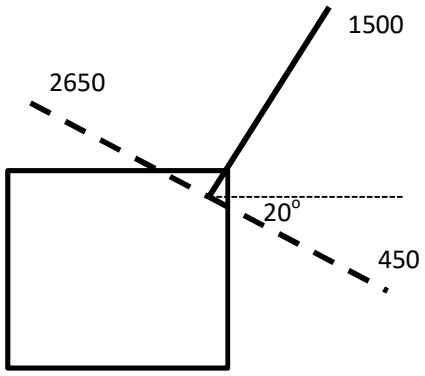


Figure 37 - Servo 2: Viewed from the right

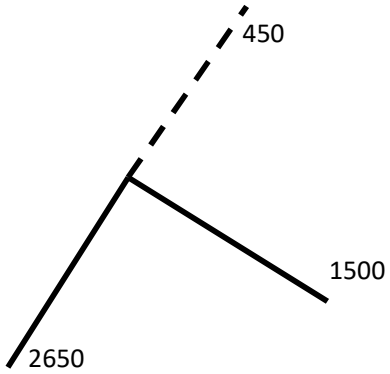


Figure 38 - Servo 3: Viewed from the right

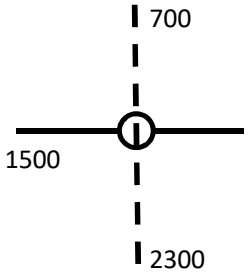


Figure 39 - Servo 4: Viewed from the gripper towards the chassis

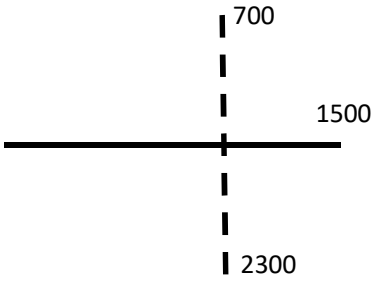


Figure 40 - Servo 5: Viewed from the right

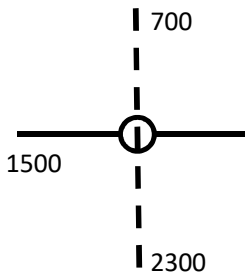


Figure 41 - Servo 6: Viewed from the gripper towards the chassis

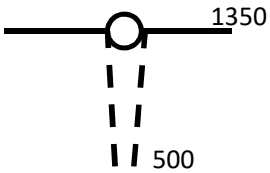


Figure 42 - Servo 7: Viewed from the top

8.2.1 Joystick control of the servos

To control the servos using the joysticks, an algorithm will be required. This algorithm will need to be able to not only convert the signals from the joysticks into a value the servos can use, but additionally do it in such a way that the arm is controllable. With this in mind, the signals from the joysticks were integrated over time. The servos were only allowed to update when a threshold integration value was reached. At this point, the integration was reset and the count restarted. This algorithm is listed below:

```
Add current joystick signal to integration

If the integration is greater than the threshold

     Add one or subtract one to the servo control signal

     Reset the integration
```

This algorithm gave the joystick controls an adequate and controllable signal from the joysticks. This algorithm was applied to all the servos when processing signals from the 4 joystick controller and only to the gripper when the IMU or HTC Vive controller signals were being processed.

8.2.2 Inverse kinematic control of the servos

Since the inverse kinematic algorithm gave an output in terms of radians of rotation, the servo signals needed to be derived from the angles provided from the inverse kinematics. Equation 8.1 through Equation 8.6 show the required conversion from inverse kinematic angle to servo signal:

$$servo_1 = 2200\left(\frac{\theta_1 + \frac{\pi}{2}}{\pi}\right) + 450 \tag{8.1}$$

$$servo_2 = 2200\left(\frac{\theta_2 + \frac{\pi}{9}}{\pi}\right) + 450 \tag{8.2}$$

$$servo_3 = 2200\left(\frac{-\theta_3 + \pi}{\pi}\right) + 450 \tag{8.3}$$

$$servo_4 = 1600\left(\frac{\theta_4 + \frac{\pi}{2}}{\pi}\right) + 700 \tag{8.4}$$

$$servo_5 = 1600\left(\frac{\theta_5 + \frac{\pi}{2}}{\pi}\right) + 700 \tag{8.5}$$

$$servo_6 = 1600\left(\frac{\theta_6 + \frac{\pi}{2}}{\pi}\right) + 700 \tag{8.6}$$

8.3 Division of Labour Between Microcontrollers

The system developed as part of this project consisted of several Arduino microcontrollers. In order to be a coherent system, the different microcontrollers were required to perform different tasks related to the project. Table 22 below shows the different tasks involved in this project and the microcontroller assigned to that task:

Table 22 - Task division by microcontroller

Task	Microcontroller
IMU reading	Arduino Nano on IMU controller
Joystick reading	Arduino Nano on joystick controller
IMU integration	Arduino Nano on IMU controller
Packet assembly	Arduino Nano on joystick and IMU controllers, PC connected to HTC Vive
Transmission	Arduino Nano on joystick and IMU controllers, Arduino Mega connected to PC
Reception	Arduino Mega in robot
Drive control	Arduino Mega in robot
Inverse kinematics	Arduino Mega in robot
Packet processing	Arduino Mega in robot
Servo control	Arduino Mega in robot
Switch processing	Arduino Mega in robot

8.4 Final system design

Figure 43 below shows the block diagram of the final system. All schematics and code is provided in the Appendices.

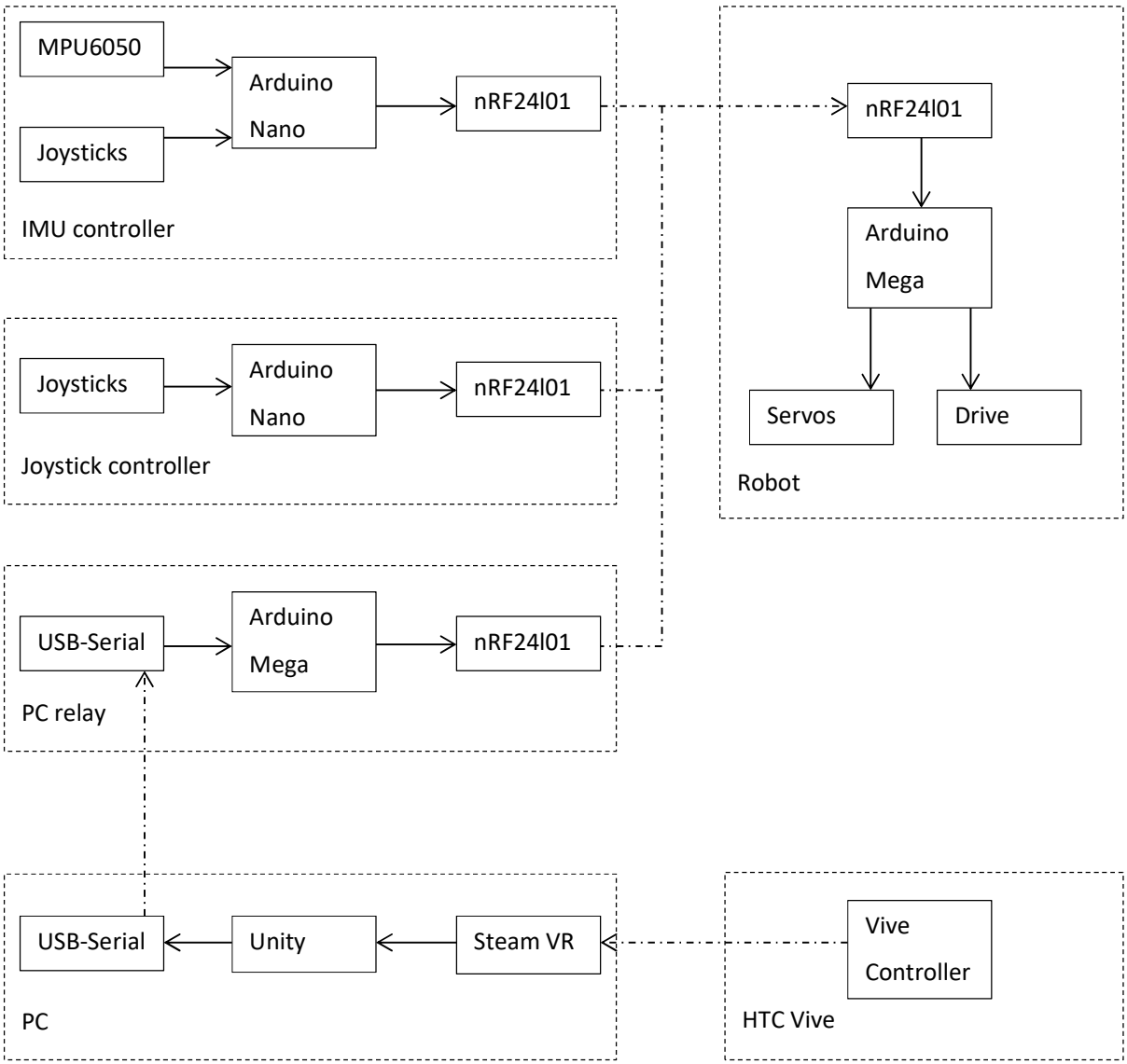


Figure 43 - System block diagram

9 Testing Results

This section provides the results of the testing performed and an analysis of those results. Some conclusions are drawn and suggestions for future research are made.

9.1 Testing Method

A testing method was required to determine the usability of the system developed in this project. This testing method is divided into 2 different parts which are listed below:

1. Quantitative testing. Results from these tests have definite numerical values to be analysed
2. Qualitative testing. Results from these tests are based on subjective experiences with the system developed.

With this in mind, a simple pick and place test was developed with the goal of providing both quantitative and qualitative data. The test took place in an arena containing all the equipment required to achieve both goals. The test was designed to eliminate as many variables as possible to single out a small number of test variables. The quantitative variables are listed below:

1. Time taken to complete the test
2. Accuracy of the placement of the object

The time taken was measured in seconds while the accuracy was measured in millimetres. Time measurements were taken with a stop watch. Accuracy was measured as the distance from the centre of the placement platform the objects final resting place was. This measurement was taken with a ruler.

The control for the testing was the joystick controller which provided a base line for comparison with the IMU controller. The HTC Vive was used as a state of the art tracking controller which provided a top level measure. Thus a comparison can be made between two separate points, the industrial standard and the state of the art. The experiment placed the IMU in a rank compared to these two points.

The arena required two platforms. Given the nature of the system, it was decided that the platforms have different heights to eliminate a user simply picking up the object and place it without changing

its elevation. The two platforms were located a distance of 1200 mm apart so that the robot was required to navigate from one platform to the other.

The object to be picked and placed was a small, 3D printed cube constructed of polylactic acid (PLA) plastic. It was 16 mm^3 in volume and had a mass of 2 g.

The robots starting location was as close to the centre of the arena as possible. The centre was marked such that the robot could easily be relocated between tests for consistency. The final location of the robot was unimportant and as such not recorded.

A total of 5 trials were performed for each controller which provided a reasonable spread of data for analysis as well as providing adequate usage for qualitative assessment.

Figure 44 below shows the layout of the arena:

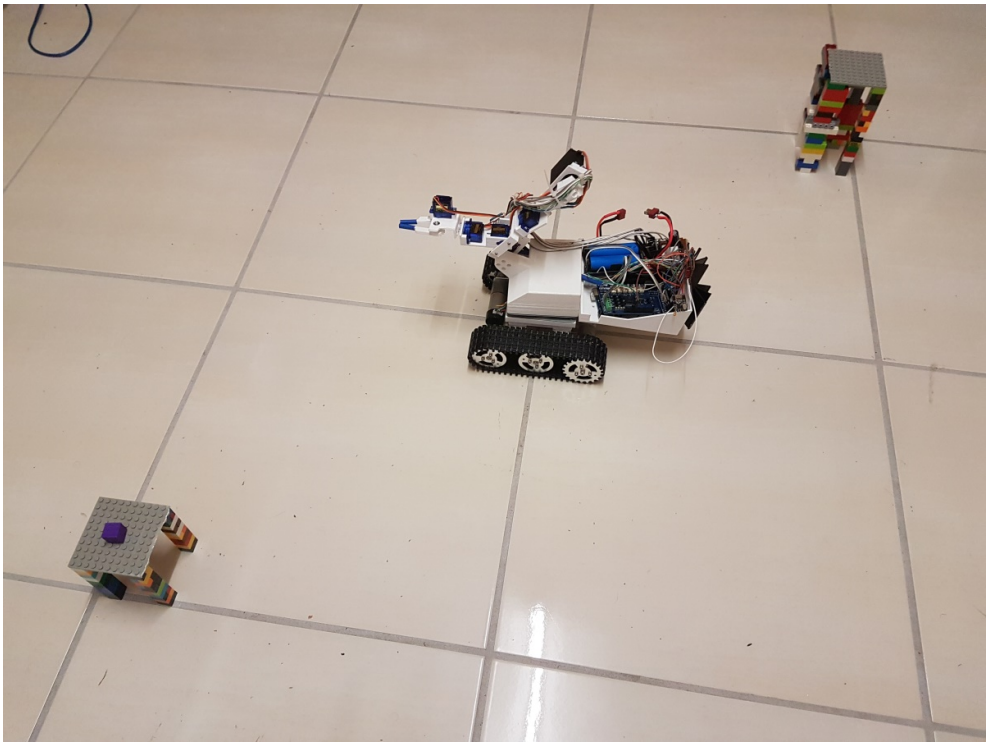


Figure 44 - Layout of the test arena

The qualitative results take the form of a narrative of how easy the controller was to use. This narrative includes the "feel" of the controller as well as an approximation of fatigue encountered while using the controller.

The obvious prediction is that the HTC Vive as the state of the art tracking controller will be both the fastest and most accurate of the controllers. The IMU is predicted to be the second fastest and accurate while the joysticks will be the slowest and least accurate.

9.2 Quantitative Results

The quantitative results have been divided into the two separate measurements as described in 9.1 Testing Method above.

9.2.1 Time results

Table 23 below shows the results for time taken to complete the test:

Table 23 - Quantitative results: Time taken

Controller	Trial	Time (s)
Joystick	1	76
	2	84
	3	45
	4	45
	5	46
Vive (track pad)	1	59
	2	50
	3	58
	4	67
	5	63
IMU	1	56
	2	53
	3	50
	4	40
	5	39
Vive (d-pad)	1	37
	2	37
	3	37
	4	35
	5	39

Figure 45 below shows these results when plotted as a column graph showing maximum, average and minimum values:

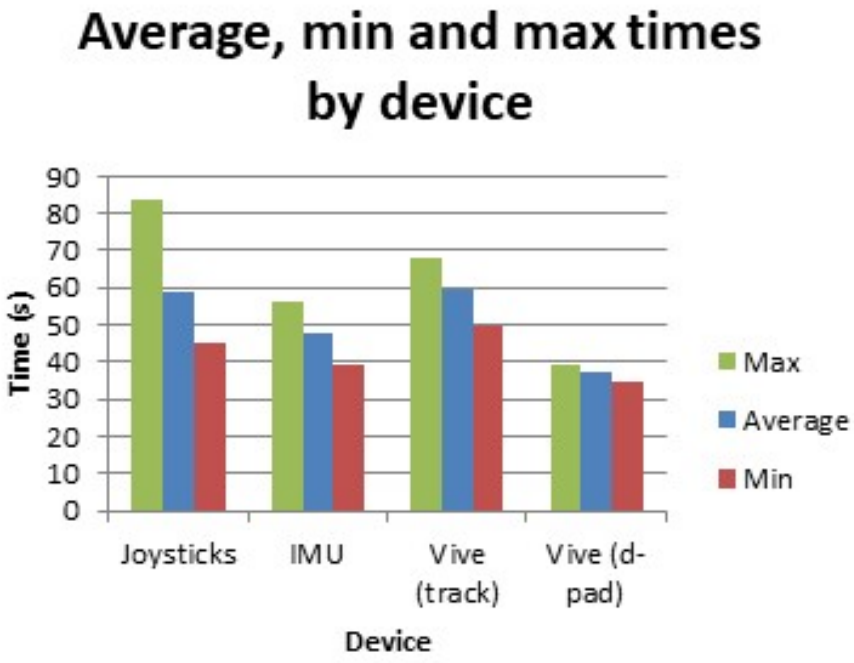


Figure 45 - Plot of time taken as maximum, minimum and average

9.2.2 Accuracy Results

Table 24 below shows the results for the accuracy of the test:

Table 24 - Quantitative results: Accuracy

Controller	Trial	Accuracy (mm)
Joystick	1	13
	2	14
	3	11
	4	45
	5	40
Vive (track pad)	1	40
	2	9
	3	16
	4	18
	5	26
IMU	1	11
	2	15
	3	2
	4	3
	5	1
Vive (d-pad)	1	47
	2	27
	3	2
	4	1
	5	3

Figure 46 below shows these results when plotted as a column graph showing maximum, average and minimum values:

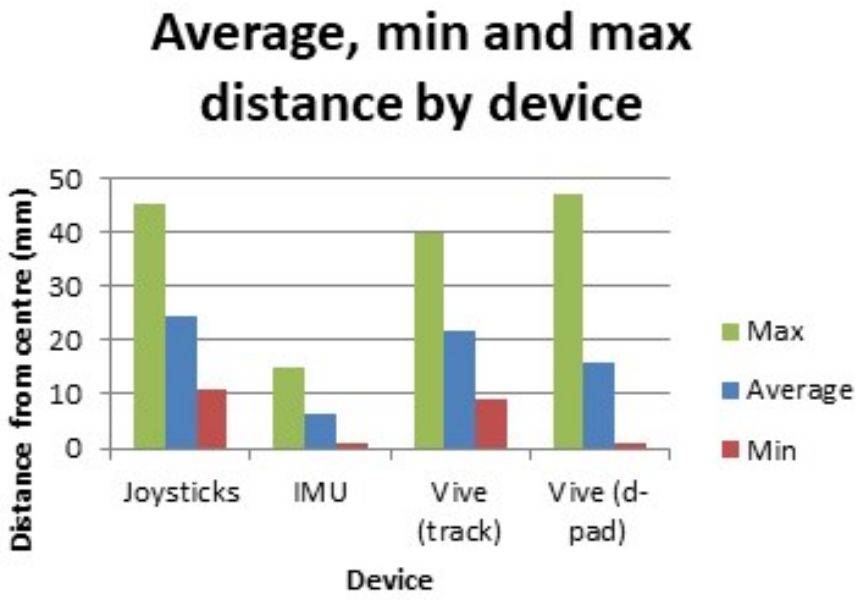


Figure 46 - Plot of accuracy taken as maximum, minimum and average

9.3 Qualitative Results

In this section, the qualitative results will be outlined. As previously mentioned, these results are the subjective experiences encountered while operating the various controllers.

9.3.1 Experiences with the joystick

The joystick controller was very cumbersome to operate. It was difficult to predict which segment of the arm that would be adjusted given the operation of any joystick axis. It was found that typically, an operation of a joystick axis would produce surprising actions in the robotic arm, an effect compounded by both inexperience with joystick controls and unfamiliarity with the rotations associated with a particular joystick movement. Despite this, the joystick to arm interface was quite smooth and provided an adequate level of accuracy.

Fatigue was not an issue when using this controller. It was found that the only seemingly contributing factor to fatigue was the concentration required to operate the correct joystick axis for a given segment of the arm. Overall the joystick controller was relatively straight forward to continuously operate.

9.3.2 Experiences with the IMU controller

The IMU controller initially felt a little cumbersome to operate however, with more experience it was found to feel like a very accurate tool to use. This is due to the discrete nature of the motions, i.e. motions occurred in bursts allowing fine adjustments to be able to be made between bursts. The discrete nature of the controller allowed some time to think and plan out the next motion.

Fatigue was definitely a factor when using this controller. Despite the fact that the controller was much lighter than the arm it was controlling, the constant gesturing in discrete steps added to fatigue. Operation of the controller felt like operation of an electronic winch with the back and forth motion required to move the arm.

9.3.3 Experiences with the HTC Vive controller

The HTC Vive controller was the least cumbersome to operate. It was found to feel the very best and most natural motion to which the robotic arm faithfully mimicked to the limits of its inverse kinematics. This could be a little distracting however as it felt like fun to operate and distracted from the task at hand. With further experience this issue would probably diminish.

Fatigue was not an issue with the HTC Vive. The controller is very light and has been designed for extended use within a virtual reality environment. The engineers at HTC have put in a lot of work to ensure that fatigue is minimised when using this controller.

9.4 Discussion of Results

The results are discussed in this section which includes a comparison of the different controllers under test with respect to the time taken and the accuracy of the controller. Additionally, a discussion of the qualitative results will be discussed.

9.4.1 Quantitative results discussion

The prediction was that the HTC Vive would be the fastest and most accurate of the controller. As discussed in section 8.1.2 the track pad method of controlling the differential drive of the robot caused issues however, these results have been included with the results from the d-pad control method.

Looking at Figure 45, it is clear that the joystick controller was the slowest method of control, followed by the HTC Vive track pad controller, followed by the IMU controller and finally the fastest was the HTC Vive with the d-pad controller. The salient points to note follow.

The minimum time for the joystick is actually lower than the minimum time for the HTC Vive using the track pad. This is likely due to the inconsistencies encountered while using the joystick method, a point reinforced when noting that the HTC Vive with track pad has a small time gap between maximum and minimum time values.

The HTC Vive using the track pad was equally as consistent as the IMU controller however, the HTC Vive using the d-pad produced the fastest and most consistent results. This is reinforced by noting the spread between the maximum and minimum values.

The prediction made at the start of this section was correct, but a surprise was encountered with the HTC Vive using the track pad. This was resolved by switching to the d-pad when using the HTC vive.

Looking at Figure 46 a surprising result is discovered in that the IMU controller was the most accurate followed by the HTC Vive using the d-pad, followed by the HTC Vive using the track pad and finally the joystick controller was the least accurate. The salient points to note follow.

Scepticism should be applied to the results of the HTC Vive using the d-pad. Noting the minimum values for the IMU and the HTC Vive using the d-pad being identical throws into question the results. When comparing the actual results in Table 24, the best three accuracies for each of these controllers are identical while the worst two are better on the IMU. Indeed the worst result for the HTC Vive with the d-pad is less accurate than the worst result from the joystick controller. The conclusion is that although this test showed that the IMU is more accurate than the HTC Vive using the d-pad, the HTC Vive certainly has the potential to be just as accurate as the IMU.

Overall, the IMU produced the most consistent accuracy results which can be explained by the discrete nature of the motions used to produce the end point for the inverse kinematics. This allowed the user more time to react to the location and plan their next move accordingly. Additionally, this would have had an impact on the time taken. The discrete nature of the motions slowed down the movement of the robot arm. Quite a profound result considering it is using only a single sensor.

This experiment provides evidence that the IMU device is capable of being as effective a control method as a HTC Vive if somewhat slower. In the case of the “average Joe”, the evidence suggests that such a device would be beneficial for these users however, the experienced user remains untested.

9.5 Future Research

The application of this technology is far reaching having possibilities in military, emergency services, disability, transport, hazardous environments and other areas too numerous to list here. Applying this technology to these services and areas will require further research.

Of most immediate requirement is the application of this technology on a mobile device such as a tablet or smart phone. Each of these devices contains an IMU device similar to the one used in this project and it would be a straight forward matter to apply this technology to the smart phone and tablet devices. The challenge would be to effectively control the mobile platform from the mobile device however, given the prevalence of touch screen technologies, this should also be straight forward.

This technology needs to be implemented on actual earth moving equipment. Such an undertaking could only be performed on small to medium excavators as the largest excavators can only effectively operate one of their joints at a time. As well as excavators and other earth moving

equipment, other hydraulic robot arms could potentially be operated using this technology such as concrete pumps and truck hoists. The possibilities are endless.

The developments with the HTC Vive controller lend themselves to the research carried out by Gleeson (2016). A fully integrated system which was not imagined by Gleeson can now be created consisting of 3D 360^{o} camera and multiple robotic arms on a moving platform all under the control of an operator using a HTC Vive. Such a system could potentially be operated over the Internet through a WiFi connection at the robot side and any other form of Internet connection at the HTC Vive side. This invention remains to be developed and tested by a future researcher.

References

Australian Communications and Media Authority, *Spectrum at 434 MHz for low powered devices,* 1999, ACMA, Australian Government, Canberra ACT, <https://www.acma.gov.au/Home/theACMA/spectrum-at-434-mhz-for-low-powered-devices>.

Australian Communications and Media Authority, *Wireless LANs in the 2.4 GHz band FAQ,* 2012, ACMA, Australian Government, Canberra ACT, <https://www.acma.gov.au/Citizen/Internet/Internet-services/Wireless-local-area-networks/wireless-lans-in-the-24-ghz-band-faqs>.

ClipDealer 2011, *Video Footage Clip - Rotation of 3D hand*, viewed 29/05/2019, <https://us.clipdealer.com/video/media/1325689>.

Common Law, *Radiocommunications (Short Range Devices) Standard 2014,* 2019, ACMA, Federal Register of Legislation, Canberra ACT, <https://www.legislation.gov.au/Details/F2019C00004>.

Espressif Systems 2019, *ESP8266EX Datasheet*, https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>.

Feng, H, Yin, C-B, Weng, W-w, Ma, W, Zhou, J-j, Jia, W-h & Zhang, Z-l 2018, 'Robotic excavator trajectory control using an improved GA based PID controller', *Mechanical Systems and Signal Processing*, vol. 105, pp. 153-68.

Gleeson, J 2016, *Remotely Operated Telepresent Robotics*, University of Southern Queensland, Queensland.

HTC Corporation 2019, *VIVE™ Australia | Discover Virtual Reality Beyond Imagination*, HTC, https://www.vive.com/au/>.

InvenSense Inc. 2013, *MPU-6000 and MPU-6050 Product Specifications*.

Kim, D, Kim, J, Lee, K, Park, C, Song, J & Kang, D 2009, 'Excavator tele-operation system using a human arm', *Automation in Construction*, vol. 18, no. 2, pp. 173-82.

Kim, I & Oh, J-H 2013, 'Inverse Kinematic Control of Humanoids under Joint Constraints', *International Journal of Advanced Robotic Systems*, vol. 10, no. 1, p. 74.

Lam, TM, Boschloo, HW, Mulder, M & van Paassen, MM 2009, 'Artificial Force Field for Haptic Feedback in UAV Teleoperation', *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 39, no. 6, pp. 1316-30.

Le, KD 2016, 'Haptic Control for Low-cost Remotely Operated Vehicles (ROVs)', University of Tasmania, Tasmania.

Lever, PJA & Wang, F-Y 1995, 'Intelligent excavator control system for lunar mining system', *Journal of Aerospace Engineering*, vol. 8, pp. 16-24.

Makers Electronics 2019, *IMU (MPU-6050)*, https://makerselectronics.com/product/imu-mpu-6050>.

Morere, Y, Hadj Abdelkader., M, A, , Cosnuau, K, Guilmois, G & Bourhis, G 2015, 'Haptic control for powered wheelchair driving assistance', *Elservier Masson*.

MuRata Manufacturing Co. Ltd. 2014, *433 MHz SAW Resonator*, https://wireless.murata.com/pub/RFM/data/ro3101.pdf>.

Nordic Semiconductor ASA 2006, *Single chip 2.4 GHz Transceiver (nRF24l01)*, https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf>.

Oh, KW, Kim, D & Hong, D 2014, 'Performance evaluation of excavator control device with EMG-based fatigue analysis', *International Journal of Precision Engineering and Manufacturing*, vol. 15, no. 2, pp. 193-9.

Standrads Australia, *Radio equipment and systems - Short rangedevices - Limits and methods of measurement,* 2017, <https://shop.standards.govt.nz/catalog/4268:2017(AS%7CNZS)/scope>.

Tafazoli, S, Salcudean, SE, Hashtrudi-Zaad, K & Lawrence, PD 2002, 'Impedance control of a teleoperated excavator', *IEEE Transactions on Control Systems Technology*, vol. 10, no. 3, pp. 355-67.

Tower Pro 2014a, *mg996r Datasheet*, http://www.towerpro.com.tw/product/mg995-robot-servo-180-rotation/>.

Tower Pro 2014b, *SG90 Datasheet*, http://www.towerpro.com.tw/product/sg90-7/>.

Valve Developer Community 2019, *SteamVR Documentation*, https://developer.valvesoftware.com/wiki/SteamVR/Environments>.

Yoon, J & Manurung, A 2010, 'Development of an intuitive user interface for a hydraulic backhoe', *Automation in Construction*, vol. 19, no. 6, pp. 779-90.

Appendix A

ENG4111/4112 Research Project

Project Specification

For: Damian Easterbrook

Title: Single Sensor Control of a Robotic Arm

Major: Mechatronic engineering

Supervisors: Tobias Low

Enrolment: ENG4111 – EXT S1, 2019 ENG4112 – EXT S2, 2019

Project Aim: To develop and evaluate the feasibility of controlling a robotic arm using a single sensor device such as an IMU

Programme: Version 1, 20th March 2019

1. Review existing controllers of robotic arms.
2. Examine existing standards of controlling robotic arms through human interface devices.
3. Review different potential methods for communication between controllers and robotic arms including radio, Blue Tooth and WiFi.
4. Design and develop a test robotic arm suitable for experimentation. Design done in Creo and 3D printed.
5. Generate suitable inverse kinematic matrices.
6. Develop various control methods including a basic IMU and HTC Vive controller. Generate protocols for tracking method (i.e. click and drag vs constant tracking, calibration).
7. Develop coding in various platforms for converting a location and rotation in space into rotation information for arm joints. This will include Arduino, Android C++ and Unity.
8. Devise a method of comparison between standard controls vs a single sensor controller.
9. Test the arm/controller combination. Compare and contrast results from tests.

If time and resources permit:

10. Test the controller in a real world situation. For example an excavator. This will require following steps 5 through 9 above.

11. Compare and contrast results from standard arm testing with results from real world situation.
12. Make recommendation on refinement of the controller and future studies in the area.

A.1 Project Timeline

Table 25 outlines the timeline for the project as a function of the phase.

Table 25 – Timeline for the project

Phase	Task
1	Equipment development and collection phase
1a	Construction of a high current power supply. For future proofing, rating will be +/- 30V, 6A variable.
1b	Accruing of missing parts such as 3.3V to 5V bidirectional digital level converters
1c	Design of robotic arm
1d	Testing and calibration of each servo motor
2	Protocol development phase
2a	Interface testing of communication devices with different microcontrollers. Arduino Uno, Arduino Mega, Arduino Nano and Raspberry Pi will be tested
2b	Signal strength testing. Determination of reliable transmission signal distance.
2c	Packet and payload development for communicating commands between modules
2d	Determination of inverse kinematic equations that govern the robotic arms including from 1c
2e	Ethical protocol considerations
3	Construction and hacking phase
3a	Construction of 2 traditional controllers. One for excavator, other for robotic arm. Master system

3b	Construction of tracking controller. One controller for both arms. Master system
3c	3D printing of parts for the robotic arm
3d	Assembly of robotic arm and chassis
3e	Disassembly of remote control excavator
3f	Actuator wiring and control signal determination within the remote control excavator
3g	Construction of slave system for excavator
3h	Construction of slave system for robotic arm
3i	Coding of protocols and inverse kinematics
3j	Initial testing to ensure everything is working safely and correctly
4	Testing phase
4a	Inform administers and participants
4b	Excavator standard control levelling test
4c	Excavator IMU tracking control levelling test
4d	Excavator HTC Vive tracking control levelling test
4e	Robotic arm standard control pick and place test
4f	Robotic arm IMU tracking control pick and place test
4g	Robotic arm HTC Vive tracking control pick and place test
4h	Collect surveys from participants
5	Data manipulation and analysis phase
5a	Prepare analysis of levelling test results. I.e. provide a comparison of speed and accuracy between the different control methods for the levelling task. Fatigue results collation.
5b	Prepare analysis of pick and place test results. I.e. provide a comparison of speed and

	accuracy between the different control methods for the pick and place test. Fatigue results collated
5c	Check the data to ensure reasonable results are being attained.
5d	Draw conclusions based on the evidence and the analysis of the evidence
6	Dissertation report phase
6a	Complete a draft version of the dissertation and submit it to the supervisor for review and feedback
6b	Attend ENG4903 to present the project, its current state, the resulting evidence and conclusions
6c	Edit the dissertation based on the feedback given by the supervisor and submit final version for marking
6d	Get dissertation published in USQ e-prints

Appendix B

B.1 Resource Requirements

Table 26 lists the resources required for the project, the price, the source and the phase the resource is used.

Table 26 - List of resources

Resource	Cost	Source	Phase
HTC Vive x 1	$1000	Student has already	2c, 3i, 4d, 4g
Remote control excavator x 1	$70	Donated by students children	3e, 3f, 3g, 3i, 3j, 4b, 4c, 4d
High current power supply x 1	$50	Most parts already owned by student. Allowed budget for other parts as required	1a, 1d, 3f
Servo motors x 6	$2 ea	eBay	1d, 3d, 3h, 3j, 4e, 4f, 4g
Communication modules x 10	$6 ea	Student has already	2a, 2b, 2c, 3a, 3b, 3g, 3h, 4b, 4c, 4d, 4e, 4f, 4g
3.3V to 5V bidirectional digital converter x 1	$20	Parts from Jaycar	1b, 2a, 2b, 2c, 3a, 3b, 3g, 3h, 3i, 3j, 4b, 4c, 4d, 4e, 4f, 4g
Arduino Uno x 8	$30 ea	Student has already	2a, 2b, 2c, 3a, 3b, 3g, 3h, 3i, 3j, 4b, 4c, 4d, 4e, 4f, 4g
Arduino Nano x 4	$30 ea	Student has already, may require more	2a, 2b, 2c, 3a, 3b, 3g, 3h, 3i, 3j, 4b,

			4c, 4d, 4e, 4f, 4g
Arduino Mega x 2	$50 ea	Student has already, may require more	2a, 2b, 2c, 3a, 3b, 3g, 3h, 3i, 3j, 4b, 4c, 4d, 4e, 4f, 4g
Raspberry Pi x 2	$75	Student has already, may require more	2a, 2b, 2c, 3a, 3b, 3g, 3h, 3i, 3j, 4b, 4c, 4d, 4e, 4f, 4g
2 axis analogue joysticks x 16	$2 ea	Student has already	3a, 3b, 3j, 4b, 4c, 4d, 4e, 4f, 4g
IMU x 1	$20	Core Electronics. Student has already	2c, 3a, 3b, 3i, 3j, 4b, 4c, 4d, 4e, 4f, 4g
Safety buttons, Red x 3	$3 ea	DIY Arcade. Student has already	3a, 3b, 3j, 4b, 4c, 4d, 4e, 4f, 4g
Robot chassis x 1	$100	Student has already	3d, 3j, 4e, 4f, 4g
18650 Li-ion batteries x 6	$13 ea	Student has already	3a, 3b, 3j, 4b, 4c, 4d, 4e, 4f, 4g
1162103 Li-ion batteries x 2	$27 ea	Student has already	3h, 3j, 4e, 4f, 4g
Charger for batteries x 1	$50	Student has already	3a, 3b, 3h, 3j, 4b, 4c, 4d, 4e, 4f, 4g
1162103 to 18650 form factor adaptor x 2	$3	Student to construct	3h, 3j, 4e, 4f, 4g
Wiring and electronics parts as needed	$100	Budget to allow for unknown factors	1b, 2a, 2b, 3a, 3b, 3c, 3d, 3e, 3f, 3g, 3h, 3i, 3j

3D printer	$250	Student has already	1a, 3a, 3b, 3c, 3g, 3h, 3j
Spool of 1.75 mm PLA plastic	$30	Student has already	1a, 3a, 3b, 3c, 3g, 3h, 3j
Galaxy S5	$299	Samsung. Student already has	2a, 2b, 2c, 4c, 4f
Android tablet	$100	Protab. Student already has	2a, 2b, 2c, 4c, 4f
Arduino development environment	Free	Arduino	2a, 2b, 2c, 2d, 3i, 3j
Unity 3D development environment	Free	Unity Technologies	2a, 2b, 2c, 3i, 3j
Steam	Free	Valve Corporation	2a, 2b, 2c, 3i, 3j, 4b, 4c, 4d, 4e, 4f, 4g
Microsoft Visual Studio	Free	Microsoft Corporation	2a, 2b, 2c, 3i, 3j
Matlab	$200	Mathworks	2d
Microsoft Office	USQ Access	Microsoft Corporation	2e, 4h, 5a, 5b, 5c, 6a, 6b, 6c
USQ Gmail	USQ Access	Google	2e, 4a, 4h, 6a, 6c
PTC Creo	USQ Access	PTC	1c, 3a, 3b, 3c, 3g, 3h, 3j

Appendix C

C.1 Risk Assessment

With project work, there comes risk. A risk is a potential outcome from an uncontrolled or unmitigated hazard. Thus, an analysis of the hazards that exist within this project is a requirement. This analysis is performed within the framework of a risk assessment. The following is the risk assessment of this project which identifies potential hazards, determines their consequences and likelihood to give a severity and finally outlines steps to be taken to minimise or remove the hazard.

C.1.1 Risk analysis

Table 27 shows the risk matrix used to determine the severity of a hazard given the likelihood and the consequence. Using this table, each potential hazard has been given a severity in Table 28

Table 27 - Risk rating matrix

	Likelihood				
Consequence	Rare	Unlikely	Possible	Very likely	Certain
Catastrophic	Moderate	Moderate	High	Critical	Critical
Major	Low	Moderate	Moderate	High	Critical
Moderate	Low	Moderate	Moderate	Moderate	High
Minor	Very low	Low	Moderate	Moderate	Moderate
Insignificant	Very low	Very low	Low	Low	Moderate

Table 28 - Hazard analysis

Hazard	Likelihood	Consequence	Severity
Electrocution	Unlikely	Catastrophic	Moderate
Burn from soldering iron	Possible	Minor	Moderate
Hot solder in eye	Unlikely	Major	Moderate
Burn from 3D printer	Possible	Minor	Moderate
Pinch injury from equipment	Unlikely	Minor	Low

Strain injury	Unlikely	Moderate	Moderate
Li-ion explosion	Rare	Major	Low

Each identified hazard requires a risk control strategy according to its severity. Table 29 lists the risk control strategies to be put in place throughout this project.

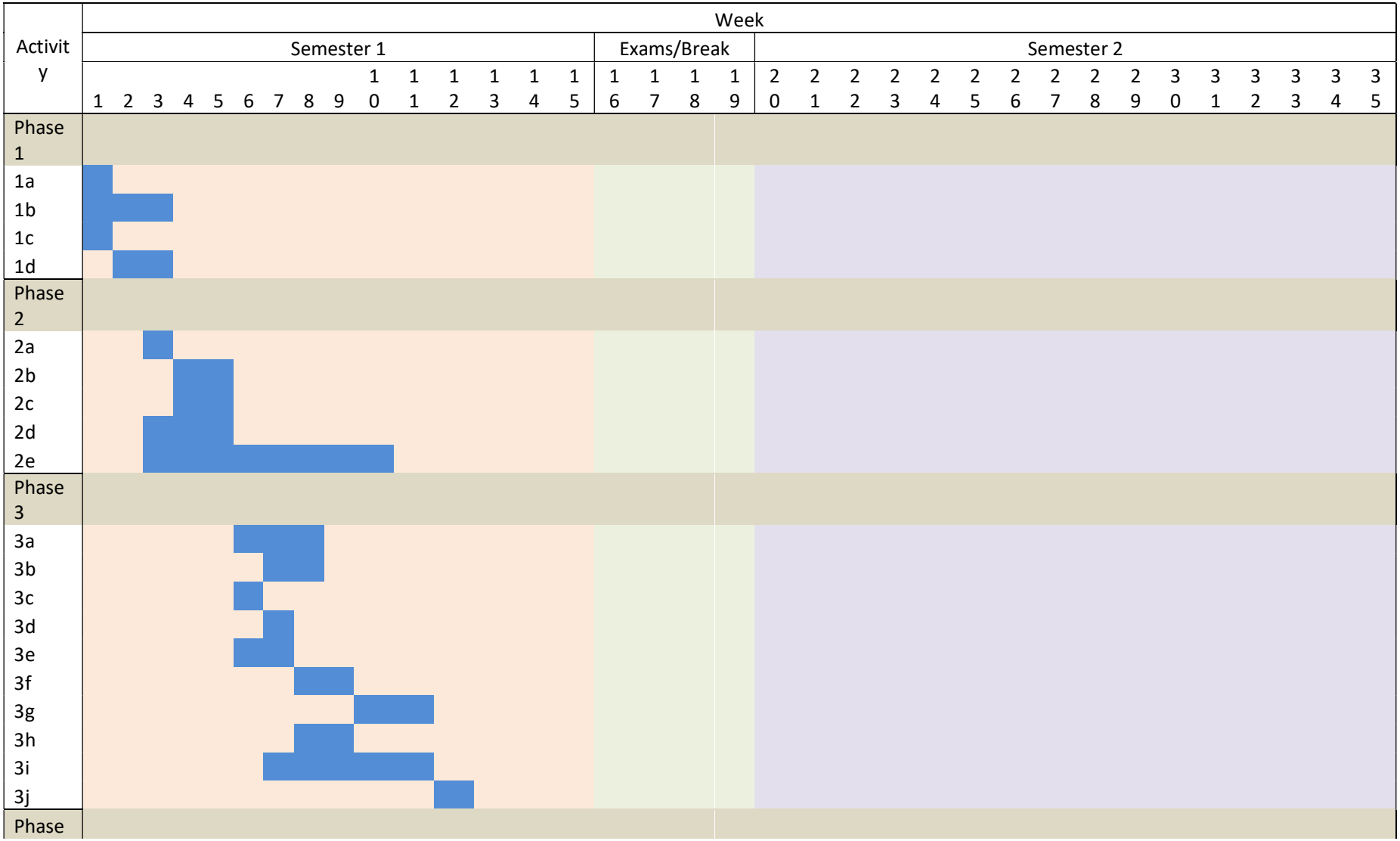
Table 29 - Risk control strategy

Risk	Strategy
Electrocution	Take extreme care when dealing with electricity. Always ensure appliances are switched off before working on them. Operate mains rated equipment with a safety switch.
Burn from soldering iron	Do not touch the hot end of the soldering iron. Use tools to handle items to be soldered
Hot solder in eye	Wear safety glasses when soldering
Burn from 3D printer	Handle hot end of 3D printer with care. Never touch the nozzle.
Pinch injury from equipment	Use care when handling robotic parts. Ensure robot is turned off before handling
Strain injury	Take routine breaks while using the various controllers
Li-ion explosion	Extreme care when soldering to the terminals of Li-ion batteries to not short them out

Appendix D

D.1 Project Schedule

The project will follow the planned schedule shown in Figure 47 below:


Activity
Week
Semester 1
Exams/Break
Semester 2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Phase 1
1a
1b
1c
1d
Phase 2
2a
2b
2c
2d
2e
Phase 3
3a
3b
3c
3d
3e
3f
3g
3h
3i
3j
Phase

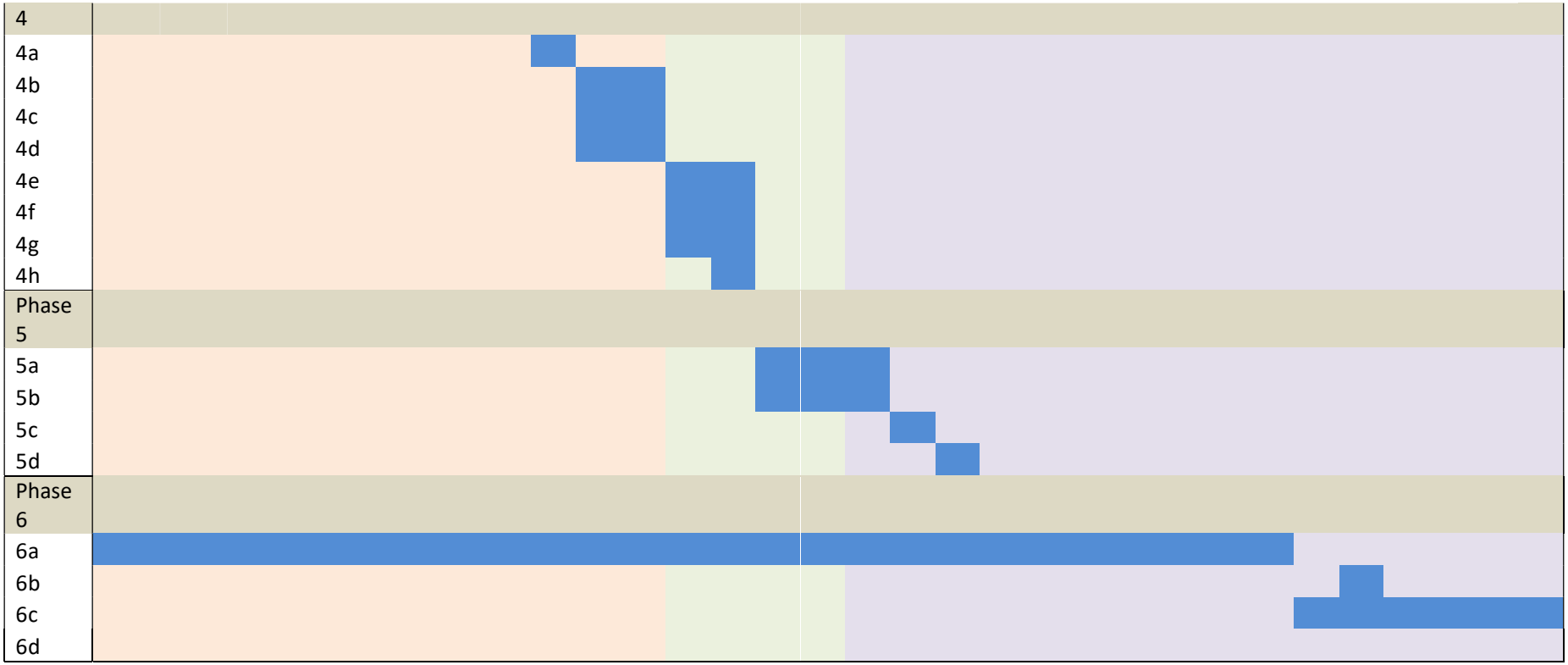
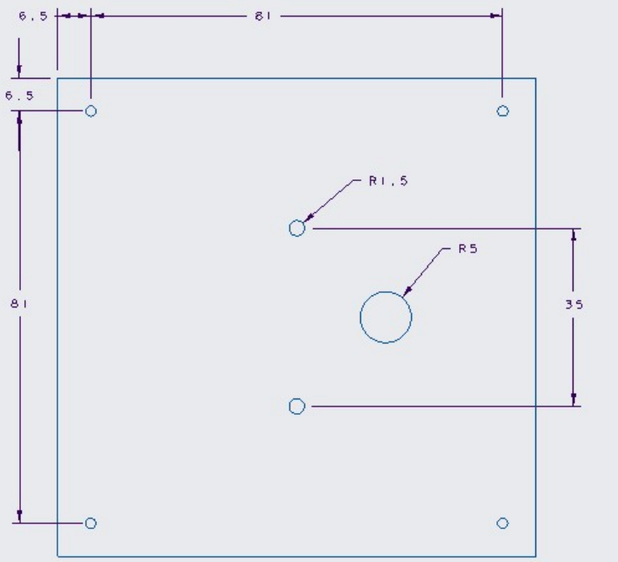
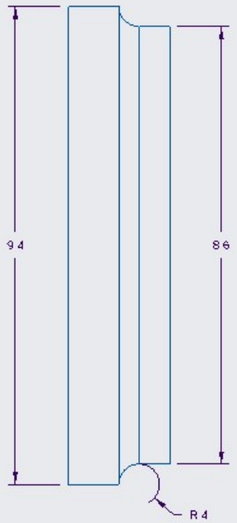


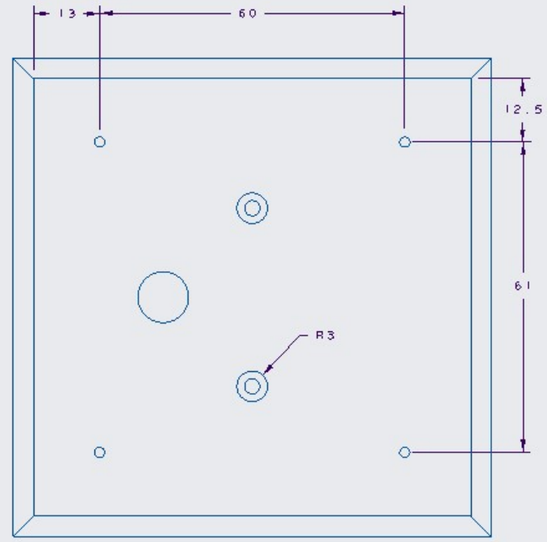
Figure 47 - Project schedule

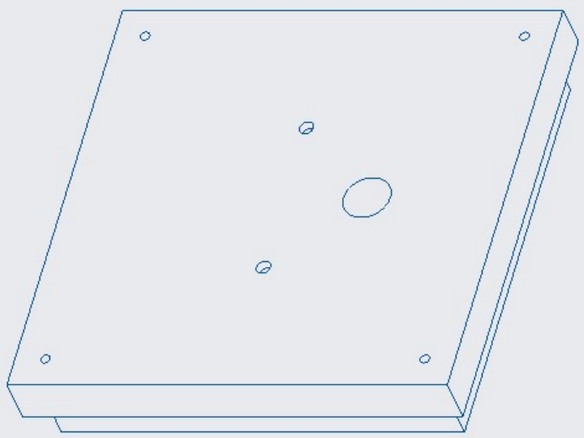
Appendix E

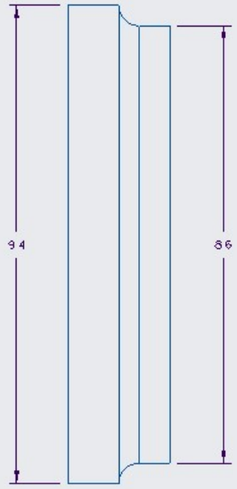
E.1 Robot and controller designs



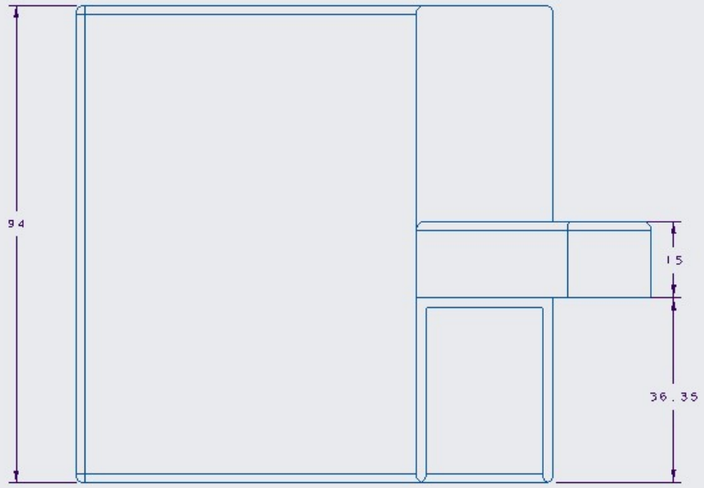


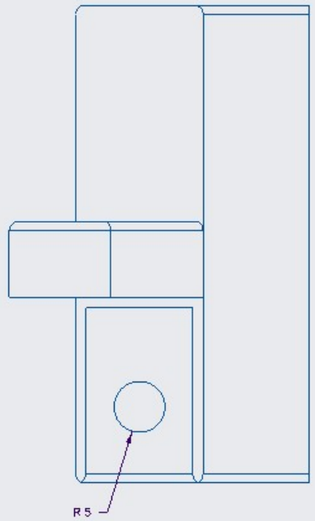


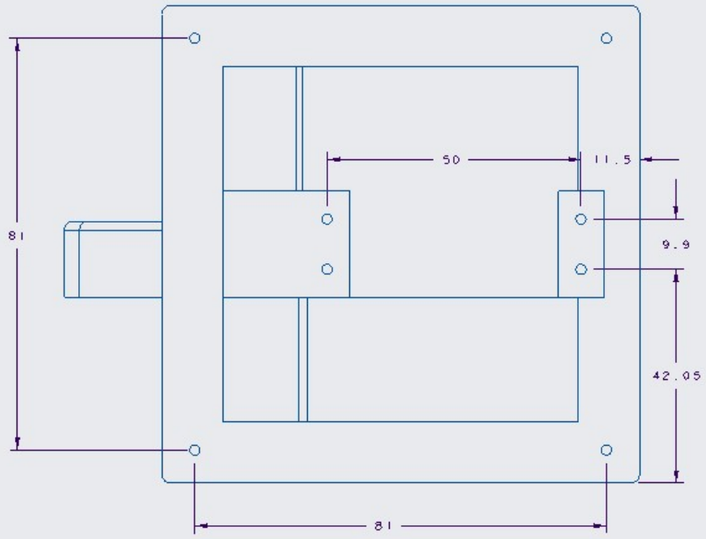


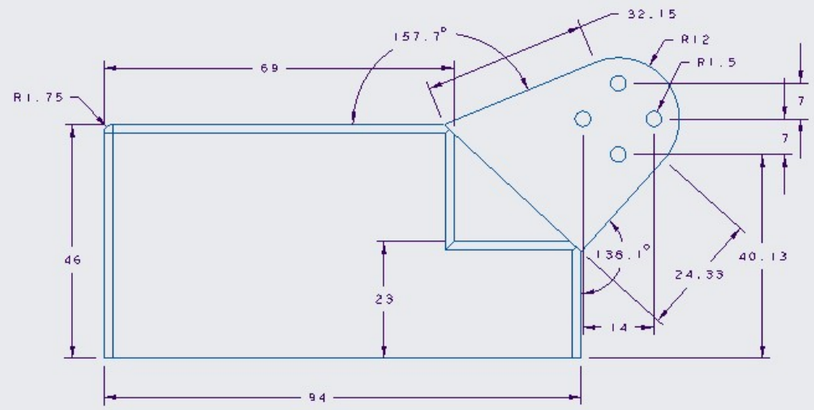


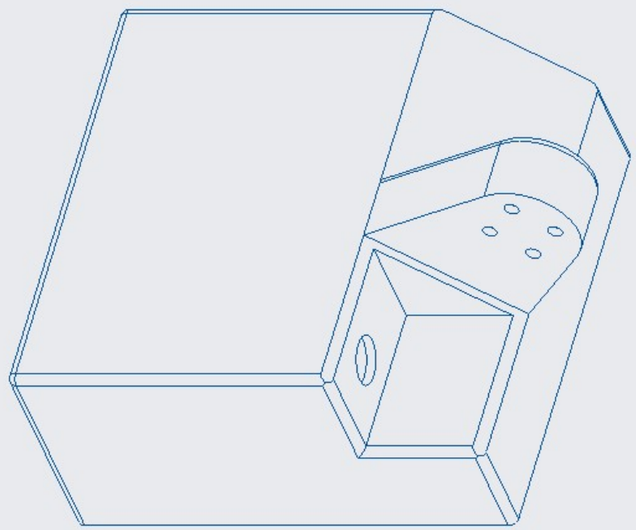
	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE Robot Body		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 1	A3
DATE: 28/02/2019			



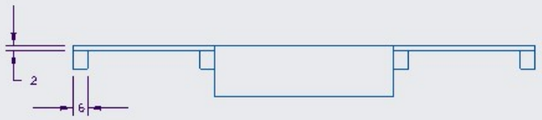


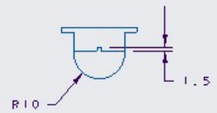


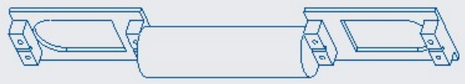




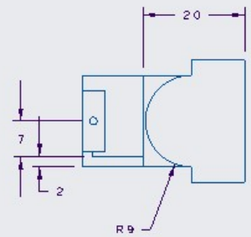
	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE: Robot Shoulder		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG. NO 2	A3
DATE: 02/03/2019			

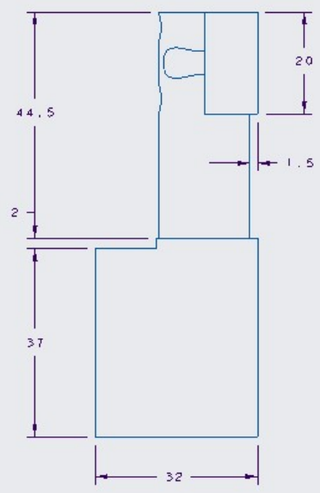


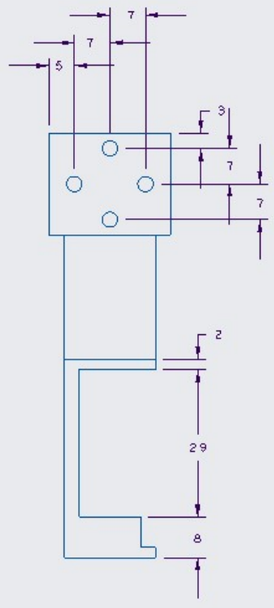


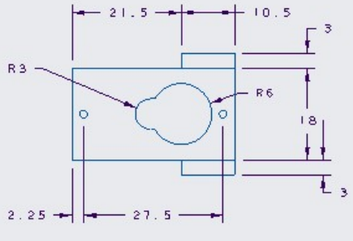


	UNIVERSITY OF SOUTHERN QUEENSLAND	
	TITLE: Robot Upper Arm	
DRAWN: Damian Easterbrook	SCALE	DRG. NO 3 / A3
DATE: 28/02/2019		

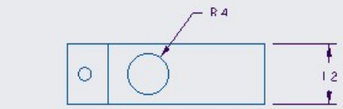


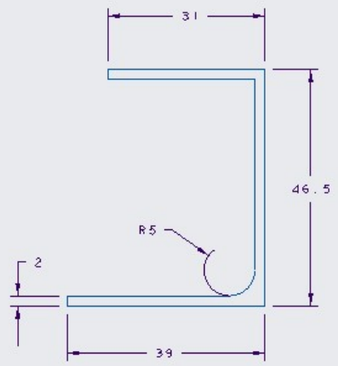


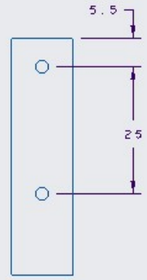


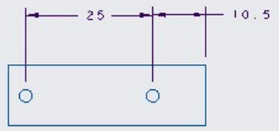


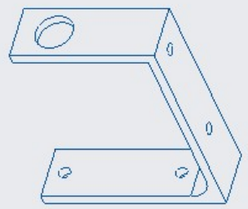
	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE Robot Elbow		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG. NO 4	A3
DATE: 28/02/2019			



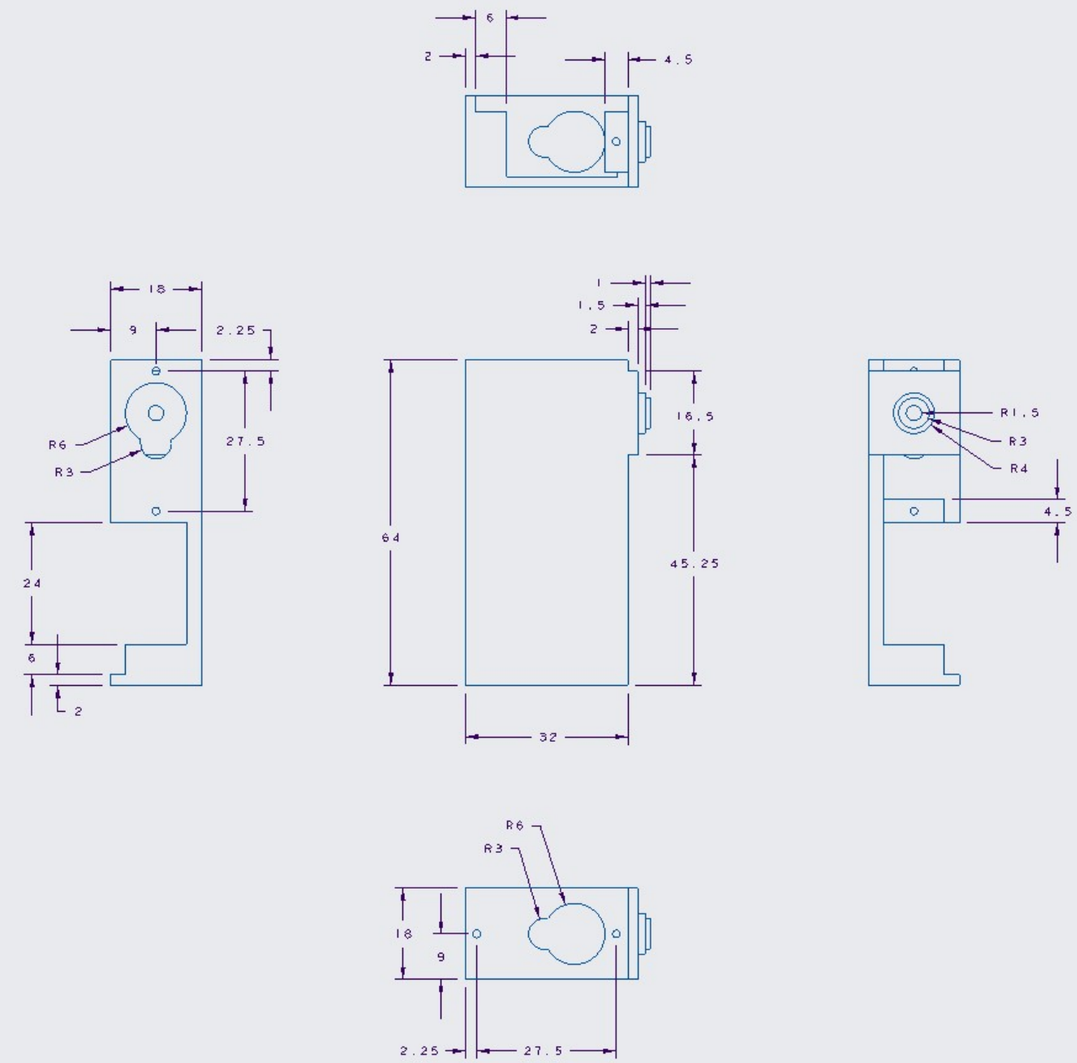




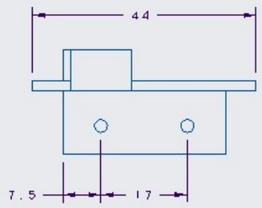


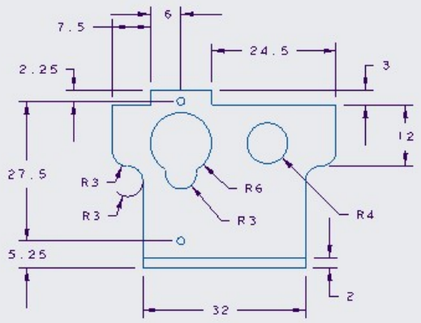


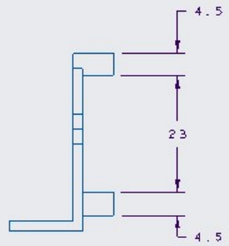
	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE: Robot Forearm		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 5	A3
DATE: 28/02/2019			

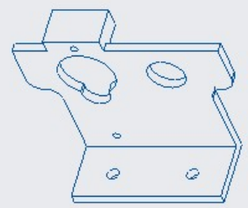


	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE: Robot Wrist		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 6	A3
DATE: 28/02/2019			

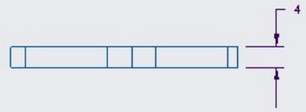


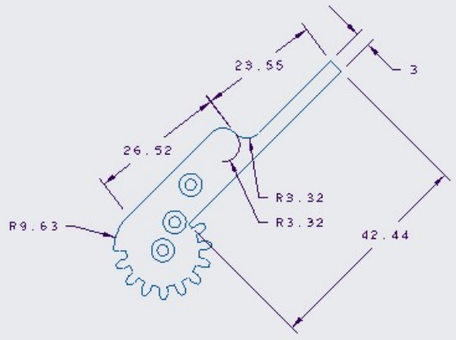




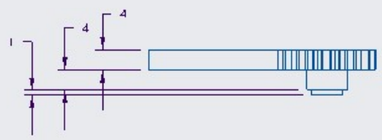


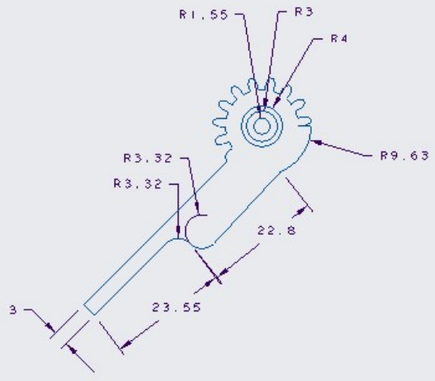
	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE: Robot Gripper Base		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 7	A3
DATE: 28/02/2019			

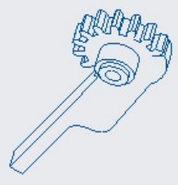




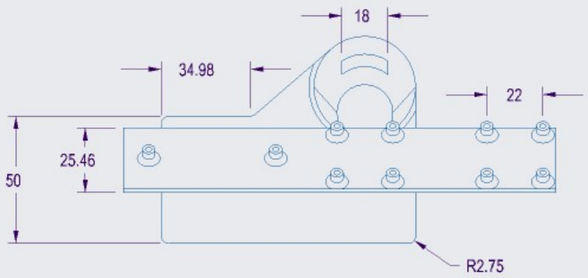
	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE Robot Powered Jaw		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 8	A3
DATE: 03/03/2019			

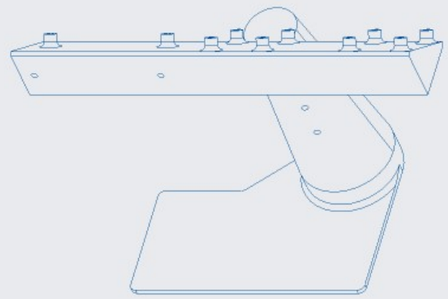


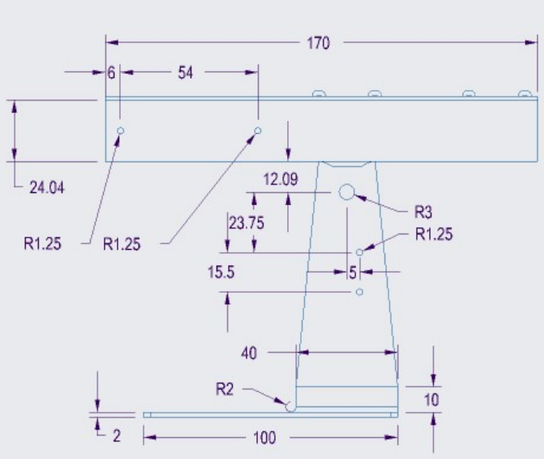




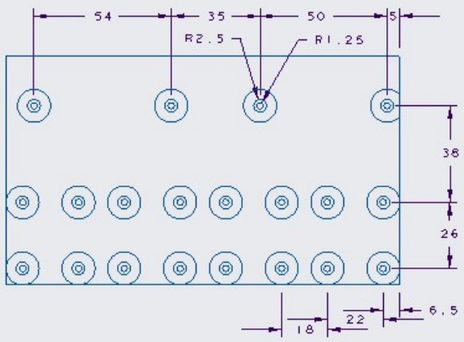
	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE Robot Loose Jaw		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 9	A3
DATE: 28/02/2019			

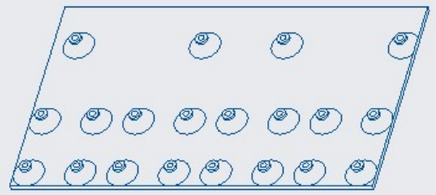


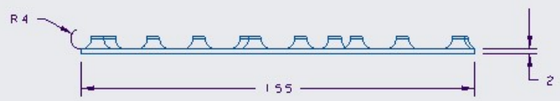


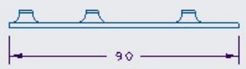


	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE IMU Enclosure		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 11	A3
DATE: 06/05/2019			







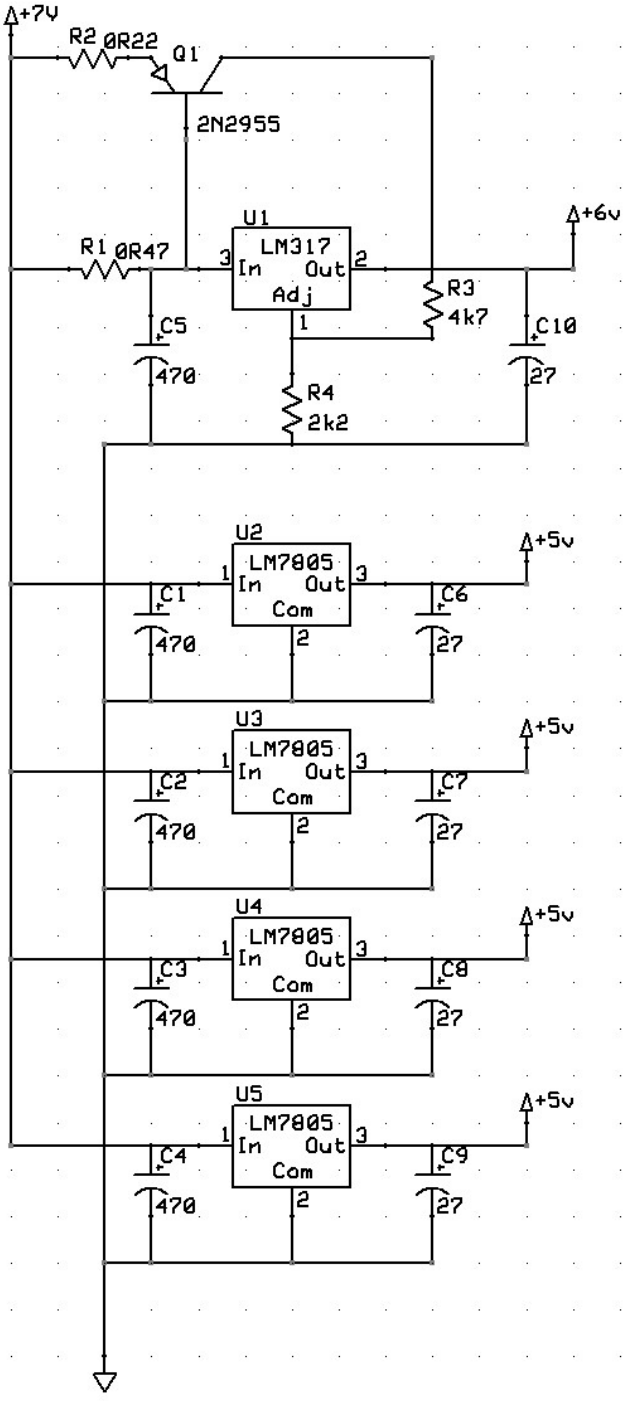


	UNIVERSITY OF SOUTHERN QUEENSLAND		
	TITLE: Joystick Controller		
DRAWN: Damian Easterbrook	SCALE 1:1	DRG.NO 12	A3
DATE: 03/05/2019			

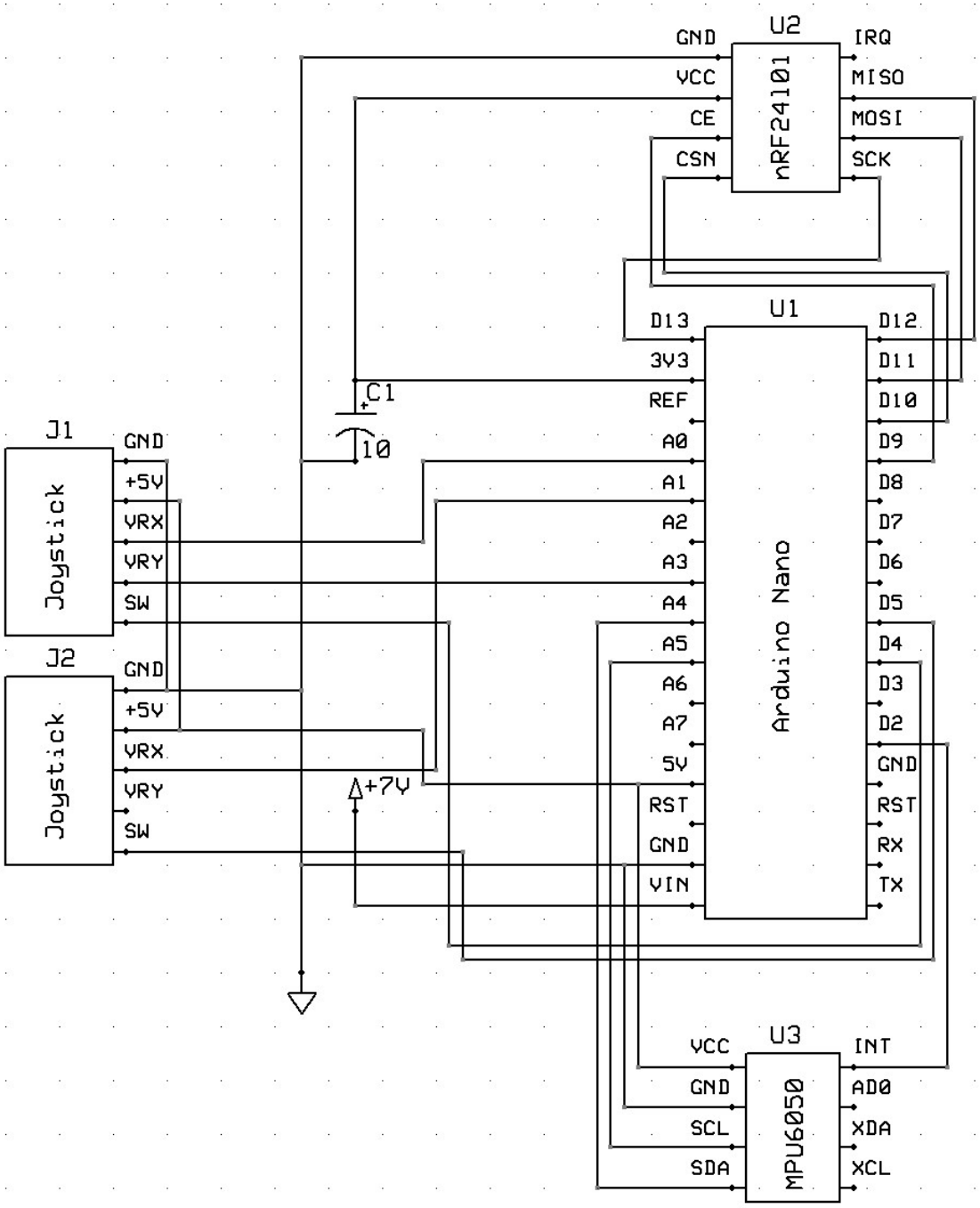
Appendix F

F.1 Circuit diagrams

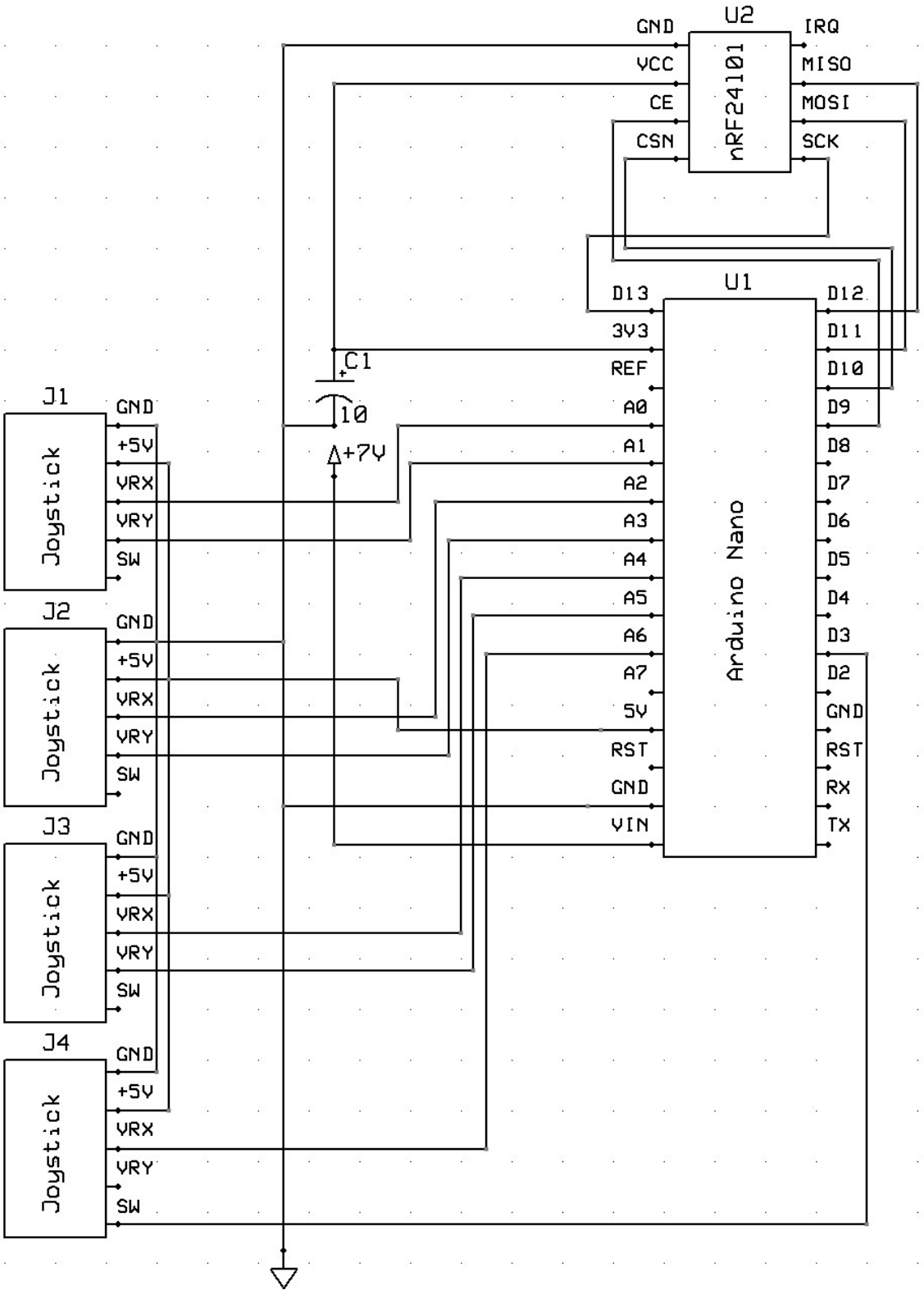
F.1.1Power circuit diagram



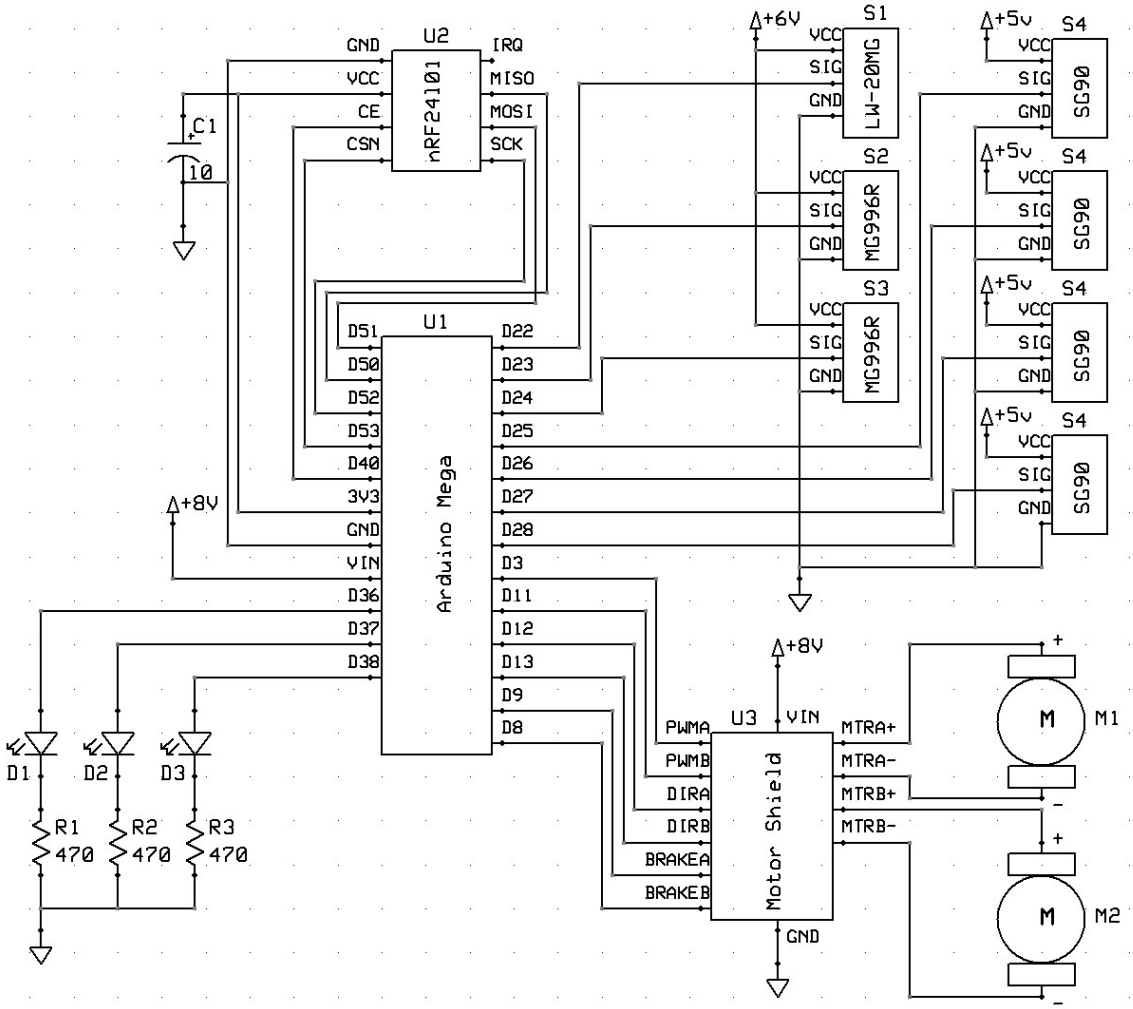
F.1.2 IMU controller circuit diagram



F.1.3 Joystick controller circuit diagram



F.1.4 Robot circuit diagram



Appendix G

G.1 Code listings

G.1.1 Joystick code

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

// Create the radio object and define the pipe to be used
RF24 radio(9, 10);
const uint32_t pipe = 0x11111111;

// Define where on the Arduino everything is connected
int joystickPins[7] = {A0, A1, A2, A3, A4, A5, A6};
int joystickVal; // temp value for joystick reading
int switchPin = 3;
int switchVal; // temp value for switch reading

// Create an array to contain the packet to be sent
uint8_t packet[9];

// Timing value for transmission
uint32_t oldTime = 0;

// SETUP FUNCTION
void setup() {
  // Serial setup for debug coms
  Serial.begin(115200);

  // Set direction of data for certain pins
  pinMode(switchPin, INPUT_PULLUP);

  // Set up radio. Channel set to 110 with maximum power
  radio.begin();
  radio.setChannel(110);
  //radio.setAddressWidth(4);
  radio.setPALevel(RF24_PA_MAX);
  radio.openWritingPipe(pipe);
  radio.stopListening();

  // Wait for radio to settle down
  delay(1000);
}

// LOOPING FUNCTION
```

```
void loop() {
  // Set packet ID
  packet[0] = 1;

  // Read the switch and write value to packet
  packet[1] = digitalRead(switchPin);

  // Loop through the analog pins and set the packet to these values
  for (int i = 0; i < 7; i++) {
    joystickVal = analogRead(joystickPins[i]) / 4;
    packet[i + 2] = joystickVal;
  }

  // Send the packet if required
  CheckAndTransmit();
}

// TRANSMISSION FUNCTION
void CheckAndTransmit() {
  // Transmit
  radio.write(packet, 9);
  Serial.println("Packet sent");
}
```

G.1.2 IMU code

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050
class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at
https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2016-04-18 - Eliminated a potential infinite loop
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo
compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read
process
//      2012-06-19 - completely rearranged DMP initialization code and
simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead
of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt
detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame
acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
```

```
//      2012-06-04 - remove accel offset clearing for better results
(thanks Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead
of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
  I2Cdev device library code is placed under the MIT license
  Copyright (c) 2012 Jeff Rowberg

  Permission is hereby granted, free of charge, to any person obtaining
a copy
  of this software and associated documentation files (the "Software"),
to deal
  in the Software without restriction, including without limitation the
rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be
included in
  all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN
  THE SOFTWARE.
  ===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the
.cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
```

```
#include "Wire.h"
#endif

#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense
evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high


#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0
= success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42
bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor
measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel
sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor
measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container

// IMU settling variables and indicator
bool settled = false;
unsigned long blinkCounter;
bool ledStatus = false;
unsigned long lastTime = 0;
unsigned long currentTime = 0;

// Set variables associating pins with hardware
const int motionPin = 5;
const int rotationPin = 4;
const int ledPin = 3;
const int joystickPins[3] = {A0, A1, A3};
```

```
// Bit wise logic values
const uint16_t MSB = 0xff00;
const uint16_t shift = 0x00ff;

// Packet construction and motion processing variables
unsigned char packet[18];
int gripVal;
int joystickVal;
int packetNum;
long v0x, v0y, v0z, v1x, v1y, v1z;
long s0x, s0y, s0z;
int rotx, roty, rotz;
VectorInt16 gyro;
int zeroCountX, zeroCountY, zeroCountZ;
VectorInt16 a0;
unsigned int scale = 512;
bool initialiseIMUVals;
Quaternion q0, qinv, qn;
long zOffset;
bool initZ = true;

// Set up the radio object and define the pipe for transmission
RF24 radio(9, 10);
const uint32_t pipe = 0x11111111;

// Debugging strings
String imuStrings[6] = {" SX: ", " SY: ", " SZ: ", " RX: ", " RY: ", "
RZ: "};


// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU
interrupt pin has gone high
void dmpDataReady() {
  mpuInterrupt = true;
}



// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
  // join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
```

```
  Wire.setClock(400000); // 400kHz I2C clock. Comment this line if
having compilation difficulties
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

  // initialize serial communication
  // (115200 chosen because it is required for Teapot Demo output, but
it's
  // really up to you depending on your project)
  Serial.begin(115200);
  while (!Serial); // wait for Leonardo enumeration, others continue
immediately

  // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or
Arduino
  // Pro Mini running at 3.3V, cannot handle this baud rate reliably due
to
  // the baud timing being too misaligned with processor ticks. You must
use
  // 38400 or slower in these cases, or use some kind of external
separate
  // crystal solution for the UART timer.

  // initialize device
  //Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
  mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
  pinMode(INTERRUPT_PIN, INPUT);

  // verify connection
  //Serial.println(F("Testing device connections..."));
  //Serial.println(mpu.testConnection() ? F("MPU6050 connection
successful") : F("MPU6050 connection failed"));

  // wait for ready
  //Serial.println(F("\nSend any character to begin DMP programming and
demo: "));
  //  while (Serial.available() && Serial.read()); // empty buffer
  //  while (!Serial.available());                 // wait for data
  //  while (Serial.available() && Serial.read()); // empty buffer again

  // load and configure the DMP
  //Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();

  // supply your own gyro offsets here, scaled for min sensitivity
  mpu.setXGyroOffset(220);
  mpu.setYGyroOffset(76);
  mpu.setZGyroOffset(-85);
  //mpu.setXAccelOffset(0);
  //mpu.setYAccelOffset(0);
```

```
  mpu.setZAccelOffset(1408); // 1688 factory default for my test chip

  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    //Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    //Serial.print(F("Enabling interrupt detection (Arduino external
interrupt "));
    //Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
    //Serial.println(F(")..."));
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady,
RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's
okay to use it
    //Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    //Serial.print(F("DMP Initialization failed (code "));
    //Serial.print(devStatus);
    //Serial.println(F(")"));
  }

  // Set up and start the radio. Channel is 110 with maximum power
  radio.begin();
  radio.setChannel(110);
  radio.setPALevel(RF24_PA_MAX);
  radio.openWritingPipe(pipe);
  radio.stopListening();

  // Reset the blink counter... used for indicating the IMU is okay and
settling
  blinkCounter = 0;
  //Serial.println("Awaiting settling time of 20 seconds");

  // Set direction for data transfer between switches and Arduino
  pinMode(motionPin, INPUT_PULLUP);
  pinMode(rotationPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);

  // Initialise motion processing variables
```

```
  s0x = 0;
  s0y = 0;
  s0z = 0;
  v0x = 0;
  v0y = 0;
  v0z = 0;
  v1x = 0;
  v1y = 0;
  v1z = 0;
  a0.x = 0;
  a0.y = 0;
  a0.z = 0;
  rotx = 0;
  roty = 0;
  rotz = 0;
  initialiseIMUVals = true;
  packetNum = 0;
}



// ================================================================
// ===                      MAIN PROGRAM LOOP                   ===
// ================================================================

void loop() {
  // Get data from the IMU
  if (!dmpReady) return;
  while (!mpuInterrupt && fifoCount < packetSize) {
    if (mpuInterrupt && fifoCount < packetSize) {
      fifoCount = mpu.getFIFOCount();
    }
  }
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

  fifoCount = mpu.getFIFOCount();
  if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) ||
fifoCount >= 1024) {
    mpu.resetFIFO();
    fifoCount = mpu.getFIFOCount();
    Serial.println(F("FIFO overflow!"));
  } else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
    mpu.getFIFOBytes(fifoBuffer, packetSize);
    fifoCount -= packetSize;
  }
  a0 = aaWorld;
  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.dmpGetEuler(euler, &q);
  mpu.dmpGetAccel(&aa, fifoBuffer);
  mpu.dmpGetGravity(&gravity, &q);
```

```
  mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
  mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
  mpu.dmpGetGyro(&gyro, fifoBuffer);

  // Need to ensure the IMU accelerometer has settled
  lastTime = currentTime;
  currentTime = micros();
  if (currentTime > 20000000) {
    ledStatus = false;
    digitalWrite(ledPin, ledStatus);
    settled = true;
  }

  // IMU is not settled so indicate that the IMU is functioning and
settling by blinking an LED
  if (!settled) {
    blinkCounter = blinkCounter + currentTime - lastTime;
    if (blinkCounter > 500000) {
      ledStatus = !ledStatus;
      digitalWrite(ledPin, ledStatus);
      blinkCounter = 0;
    }
  }

  // IMU has settled
  if (settled) {
    // Get a value for a z offset to correct discrepency from gravity
    if(initZ){
      initZ = false;
      zOffset = aaWorld.z;
    }

    // Is the motion capture switch pressed
    if (!digitalRead(motionPin)) {
      // Initialise values if needed
      if (initialiseIMUVals) {
        s0x = 0;
        s0y = 0;
        s0z = 0;
        v0x = 0;
        v0y = 0;
        v0z = 0;
        v1x = 0;
        v1y = 0;
        v1z = 0;
        a0.x = 0;
        a0.y = 0;
        a0.z = 0;
        initialiseIMUVals = false;
      }
      digitalWrite(ledPin, HIGH);
      // Integrate velocity
```

```
      v1x = v0x;
      v1y = v0y;
      v1z = v0z;
      v0x = v0x + (aaWorld.x + a0.x) / 512;
      v0y = v0y + (aaWorld.y + a0.y) / 512;
      v0z = v0z + (aaWorld.z - z0ffset + a0.z) / 512;

      // Constrict velocity if needed
      if (v1x == v0x) {
        zeroCountX++;
      } else {
        zeroCountX = 0;
      }
      if (v1y == v0y) {
        zeroCountY++;
      } else {
        zeroCountY = 0;
      }
      if (v1z == v0z) {
        zeroCountZ++;
      } else {
        zeroCountZ = 0;
      }
      if (zeroCountX > 10) {
        v0x = 0;
      }
      if (zeroCountY > 10) {
        v0y = 0;
      }
      if (zeroCountZ > 10) {
        v0z = 0;
      }

      // Integrate displacement
      s0x = s0x + (v0x + v1x) / 2;
      s0y = s0y + (v0y + v1y) / 2;
      s0z = s0z + (v0z + v1z) / 2;
    } else {
      // Is the rotation capture switch pressed
      if (!digitalRead(rotationPin)) {
        // Initialise values if needed... note that we initialise the
quaternion to the current orientation
        // this is so we have a frame of reference by which to calculate
orientation
        if (initialiseIMUVals) {
          q0 = q;
          initialiseIMUVals = false;
        }
        digitalWrite(ledPin, HIGH);
        // Invert the orientation quaternion
        qinv.w = q.w;
        qinv.x = -q.x;
```

```
        qinv.y = -q.y;
        qinv.z = -q.z;

        // This operation extracts the Euler angles as well as a
rotation about that axis
        // Only the Euler angles will be transmitted
        qn = QMultiply(qinv, q0);

        // Prepare angles for sending
        roty = (int)(asin(qn.x) * 3600 / PI);
        rotz = (int)(asin(qn.y) * 3600 / PI);
        rotx = (int)(asin(qn.z) * 3600 / PI);
        Serial.print(rotx);
        Serial.print(' ');
        Serial.print(roty);
        Serial.print(' ');
        Serial.print(rotz);
        Serial.println(' ');
      } else {
        initialiseIMUVals = true;
      }
    }
    // Set the packet ID
    packet[0] = 0x02;

    // Read the state of the buttons and write to the packet
    packet[1] = digitalRead(motionPin);
    packet[2] = digitalRead(rotationPin);

    // Read the positions of the joysticks and write to the packet
    for (int i = 0; i < 3; i++) {
      joystickVal = analogRead(joystickPins[i]);
      packet[i + 3] = joystickVal / 4;
    }

    // Convert the 16 bit values of position and orientation into 2 8
bit numbers
    packet[6] = (s0x & MSB) >> 8;
    packet[7] = s0x & shift;
    packet[8] = (s0y & MSB) >> 8;
    packet[9] = s0y & shift;
    packet[10] = (s0z & MSB) >> 8;
    packet[11] = s0z & shift;
    packet[12] = (rotx & MSB) >> 8;
    packet[13] = rotx & shift;
    packet[14] = (roty & MSB) >> 8;
    packet[15] = roty & shift;
    packet[16] = (rotz & MSB) >> 8;
    packet[17] = rotz & shift;
    //PrintPacket();

    // Try to send the packet
```

```
    SendPacket();
  }
}


// TRANSMISSION FUNCTION
void SendPacket() {
  //Serial.write(packet, 38);
  radio.write(packet, 18);
}

// PACKET DEBUGGING FUNCTION
void PrintPacket() {
  Serial.print("I: ");
  Serial.print(packet[0]);
  Serial.print(" R: ");
  Serial.print(packet[1]);
  Serial.print(" M: ");
  Serial.print(packet[2]);
  Serial.print(" ");
  for (int i = 0; i < 3; i++) {
    Serial.print(i);
    Serial.print(": ");
    joystickVal = packet[i + 3];
    Serial.print(joystickVal);
    Serial.print(" ");
  }
  for (int i = 0; i < 6; i++) {
    Serial.print(imuStrings[i]);
    long val = packet[i * 2 + 7] << 8 | packet[i * 2 + 8];
    Serial.print(val);
  }
  Serial.println(" ");
}

// QUATERNION MULTIPLYING FUNCTION
Quaternion QMultiply(Quaternion q1, Quaternion q2) {
  Quaternion result;
  result.w = q1.w * q2.w - q1.x * q2.x - q1.y * q2.y - q1.z * q2.z;
  result.x = q1.w * q2.x + q1.x * q2.w + q1.y * q2.z - q1.z * q2.y;
  result.y = q1.w * q2.y - q1.x * q2.z + q1.y * q2.w + q1.z * q2.x;
  result.z = q1.w * q2.z + q1.x * q2.y - q1.y * q2.x + q1.z * q2.w;
  return result;
}
```

G1.1.3 HTC Vive radio forwarding

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
```

```
//#include <SoftwareSerial.h>

RF24 radio(40, 53);
const uint32_t pipe = 0x11111111;
//SoftwareSerial sp(13,12);

uint8_t readPacket[64];
uint8_t packet[18] = {0x02, 0x01, 0x01, 0x7f, 0x7f, 0x7f, 0x01, 0x00,
0x01, 0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00};
bool canSend = true;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial3.begin(4800);
  //sp.begin(38400);
  radio.begin();
  radio.setChannel(110);
  radio.setPALevel(RF24_PA_MAX);
  radio.openWritingPipe(pipe);
  radio.stopListening();
  delay(1000);
  radio.write(packet, 18);
}

void loop() {
  // put your main code here, to run repeatedly:
  //radio.write(packet, 18);
  //delay(1);

  if (Serial3.available()) {
    Serial.print(Serial3.available());
    Serial.print(' ');
    Serial3.readBytes(packet, 18);
    radio.write(packet, 18);
    delay(5);
    for (int i = 0; i < 18; i++) {
      Serial.print(packet[i]);
      Serial.print(' ');
    }
    Serial.println(' ');
  }
}
```

G.1.4 Robot code

```
#include <SPI.h>
#include "nRF24L01.h"
```

```
#include "RF24.h"
#include <Servo.h>

// Define the packet array
unsigned char packet[34];

// Construct the radio object and define a pipe
RF24 radio(40, 53);
const uint32_t pipe = 0x11111111;

// Debugging variables
int joystickVal;
String imuStrings[6] = {" SX: ", " SY: ", " SZ: ", " RX: ", " RY: ", "
RZ: "};

// Definitions of the different pins devices are attached to
int pwmAPin = 3;
int pwmBPin = 11;
int dirAPin = 12;
int dirBPin = 13;
int brakeAPin = 9;
int brakeBPin = 8;
int led1Pin = 36;
int led2Pin = 37;
int led3Pin = 38;
int servoPins[7] = {22, 23, 24, 25, 26, 27, 28};

// Construct the servo objects and define limits, positions, directions
and mappings
Servo servos[7];
int servoLowerLimits[7] = {450, 450, 450, 700, 700, 700, 500};
int servoUpperLimits[7] = {2650, 2650, 2650, 2300, 2300, 2300, 1350};
int servoValues[7] = {1500, 2500, 2200, 1500, 1500, 1500, 1350};
int servoSubValues[7] = {0, 0, 0, 0, 0, 0, 0};
int servoMap[7] = {1, 0, 2, 3, 4, 5, 6};
int servoDir[7] = {1, 1, -1, -1, -1, -1, 1};

// IK variables: lengths of arm segments
float h = 132.13;
float s1 = 54.3;
float s2 = 150.0;
float s3 = 78.5;
float s4 = 23.0;
float s5 = 58.5;
float t = 65.0;

// IK variables: position and orientation
float Lx = 250.0, Ly = 0.0, Lz = 250.0;
float Ow = 0.0, Ox = 1.0, Oy = 0.0, Oz = 0.0;

// IK variables: place holders
float oldLx, oldLy, oldLz;
```

```
float oldOw, oldOx, oldOy, oldOz;
bool hasSet = false;

// IK variables: theta values for angles in the arms
float t1, t2, t3, t4, t5, t6;

// Quaternion structure
struct Quaternion {
  float w;
  float x;
  float y;
  float z;
};

// Robots current state
int state = 0;

// Debugging variable
int iteration = 0;

// Button press register
bool hasButton = false;

// SETUP FUNCTION
void setup(void) {
  // Begin serial for debugging
  Serial.begin(115200);

  // Set up radio. Channel is 110. Receiver only so transmission power
not set
  radio.begin();
  radio.setChannel(110);
  //radio.setAddressWidth(4);
  //radio.setPALevel(RF24_PA_MIN);
  radio.openReadingPipe(1, pipe);
  radio.startListening();

  // Set direction for data between Arduino and devices
  pinMode(pwmAPin, OUTPUT);
  pinMode(pwmBPin, OUTPUT);
  pinMode(dirAPin, OUTPUT);
  pinMode(dirBPin, OUTPUT);
  pinMode(brakeAPin, OUTPUT);
  pinMode(brakeBPin, OUTPUT);
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(led3Pin, OUTPUT);

  // Calculate initial position and orientation
  IK();
  ConvertAngles();
```

```
  // Set up the servo motors
  for (int i = 0; i < 7; i++) {
    pinMode(servoPins[i], OUTPUT);
    servos[i].attach(servoPins[i], servoLowerLimits[i],
servoUpperLimits[i]);
    servos[i].writeMicroseconds(servoValues[i]);
  }
  delay(1000);
}

// LOOPING FUNCTION
void loop(void) {
  //bool goodSignal = radio.testRPD();

  // Read the radio and indicate if a signal is present
  if (radio.available(pipe)) {
    //Serial.println(goodSignal ? "Strong signal" : "Weak signal");
    bool done = false;
    while (!done) {
      done = radio.read(packet, 34);
      digitalWrite(led1Pin, 1);
      //PrintPacket();
    }
  } else {
    digitalWrite(led1Pin, 0);
  }

  // Check for a valid packet ID
  if (packet[0] > 0) {
    // Check if the state needs to be changed
    CheckChangeState();

    // Decide what to do depending on the state
    switch (state) {
      case -1:
        StraightLine();
        break;
      case 0:
        WheelDrive();
        if (packet[0] == 2) {
          JawDrive();
        }
        digitalWrite(led2Pin, 0);
        break;
      case 1:
        ArmDriveJS();
        JawDrive();
        digitalWrite(led2Pin, 1);
        break;
      case 2:
        ArmDriveIMU();
        digitalWrite(led2Pin, 1);
```

```
        break;
    }
  }

  // Move the servos
  for (int i = 0; i < 7; i++) {
    servos[i].writeMicroseconds(servoValues[i]);
  }
}

void StraightLine() {
  Serial.print("Hello");
  if (iteration < 500) {
    Lz = Lz + 0.1;
    IK();
    ConvertAngles();
    iteration++;
  } else {
    if (iteration >= 500 && iteration < 1500) {
      Lz = Lz - 0.1;
      IK();
      ConvertAngles();
      iteration++;
    } else {
      if (iteration >= 1500 && iteration < 2000) {
        Lz = Lz + 0.1;
        IK();
        ConvertAngles();
        iteration++;
      } else {
        iteration = 0;
      }
    }
  }
}

// DRIVING THE WHEELS
void WheelDrive() {
  // Define variables for wheel drive processing
  int driveA, driveB, dirA, dirB, brakeA = 0, brakeB = 0;
  int rawA, rawB;

  // Check the ID of the packet and extract the drive data appropriately
  if (packet[0] == 1) {
    rawA = packet[2];
    rawB = packet[8];
  }
  if (packet[0] == 2) {
    rawA = packet[3];
    rawB = packet[4];
  }
  rawA -= 127;
```

```
  rawB -= 127;
  if (rawA > 40) {
    driveA = rawA * 2 - 1;
    dirA = 0;
  } else {
    if (rawA < -40) {
      driveA = -rawA * 2 + 1;
      dirA = 1;
    } else {
      driveA = 0;
      dirA = 0;
      brakeA = 1;
    }
  }
  if (rawB > 40) {
    driveB = rawB * 2 - 1;
    dirB = 0;
  } else {
    if (rawB < -40) {
      driveB = -rawB * 2 + 1;
      dirB = 1;
    } else {
      driveB = 0;
      dirB = 0;
      brakeB = 1;
    }
  }
  // Drive the motors
  digitalWrite(dirAPin, dirA);
  digitalWrite(dirBPin, dirB);
  analogWrite(pwmAPin, driveA);
  analogWrite(pwmBPin, driveB);
  digitalWrite(brakeAPin, brakeA);
  digitalWrite(brakeBPin, brakeB);
}

// ARM CONTROL USING JOYSTICKS
void ArmDriveJS() {
  // Temp storage of arm values
  int armValues[7];

  // Get arm values from packet
  for (int i = 0; i < 6; i++) {
    armValues[servoMap[i]] = packet[i + 2];
  }

  // Calculate and limit arm values
  for (int i = 0; i < 6; i++) {
    armValues[i] -= 127;
    if (armValues[i] < 25 && armValues[i] > -25) {
      armValues[i] = 0;
    }
```

```
    servoSubValues[i] += armValues[i];
    if (servoSubValues[i] > 1028) {
      servoValues[i] += servoDir[i];
      servoSubValues[i] -= 1028;
    }
    if (servoSubValues[i] < -1028) {
      servoValues[i] -= servoDir[i];
      servoSubValues[i] += 1028;
    }
    if (servoValues[i] > servoUpperLimits[i]) {
      servoValues[i] = servoUpperLimits[i];
    }
    if (servoValues[i] < servoLowerLimits[i]) {
      servoValues[i] = servoLowerLimits[i];
    }
  }
}

// ARM CONTROL USING IMU OR HTC VIVE
void ArmDriveIMU() {
  // Extract data from packet
  int s0x = (packet[6] << 8) + packet[7];
  int s0y = (packet[8] << 8) + packet[9];
  int s0z = (packet[10] << 8) + packet[11];
  int rotx = (packet[12] << 8) + packet[13];
  int roty = (packet[14] << 8) + packet[15];
  int rotz = (packet[16] << 8) + packet[17];
  Quaternion qy, qz, q, qi, qv;

  // Calculate end location
  if (packet[1] == 0) {
    Lx = oldLx + (float)s0x / 350;
    Ly = oldLy + (float)s0y / 350;
    Lz = oldLz + (float)s0z / 350;
  }

  // Calculate end orientation
  if (packet[2] == 0) {
    Ow = oldOw + (float)rotx * PI / 3600;
    qy.w = cos((float)roty * PI / 7200);
    qy.x = 0;
    qy.y = sin((float)roty * PI / 7200);
    qy.z = 0;
    qz.w = cos((float)rotz * PI / 7200);
    qz.x = 0;
    qz.y = 0;
    qz.z = sin((float)rotz * PI / 7200);
    qv.w = 0;
    qv.x = 1;
    qv.y = 0;
    qv.z = 0;
    q = QMultiply(qz, qy);
```

```
    qi.w = q.w;
    qi.x = -q.x;
    qi.y = -q.y;
    qi.z = -q.z;
    qv = QMultiply(q, qv);
    qv = QMultiply(qv, qi);
    Ox = qv.x;
    Oy = qv.y;
    Oz = qv.z;
  }

  // Calculate IK
  IK();
  ConvertAngles();

  // Kill the motors
  digitalWrite(pwmAPin, 0);
  digitalWrite(pwmBPin, 0);
  digitalWrite(brakeAPin, 1);
  digitalWrite(brakeBPin, 1);
}

// DRIVE THE JAW SERVO
void JawDrive() {
  // Temp storage of jaw value
  int raw;

  // Extract value from packet
  if (packet[0] == 1) {
    raw = packet[8];
  }
  if (packet[0] == 2) {
    raw = packet[5];
  }

  // Calculate, limit and write value to servo
  raw -= 127;
  if (raw < 25 && raw > -25) {
    raw = 0;
  }
  servoSubValues[6] += raw;
  if (servoSubValues[6] > 1028) {
    servoValues[6]++;
    servoSubValues[6] = 0;
  }
  if (servoSubValues[6] < -1028) {
    servoValues[6]--;
    servoSubValues[6] = 0;
  }
  if (servoValues[6] > servoUpperLimits[6]) {
    servoValues[6] = servoUpperLimits[6];
  }
```

```
  if (servoValues[6] < servoLowerLimits[6]) {
    servoValues[6] = servoLowerLimits[6];
  }
}

// CHANGE CONTROL FROM DRIVE MOTORS TO SERVOS AND BACK
void CheckChangeState() {
  // What packet is being used?
  if (packet[0] == 1) {
    // Joystick control. Check the state of the button and change the
state
    if (state == 0) {
      if (!hasButton && packet[1] == 0) {
        hasButton = true;
        state = 1;
      }
      if (hasButton && packet[1] == 1) {
        hasButton = false;
      }
    } else {
      if (!hasButton && packet[1] == 0) {
        hasButton = true;
        state = 0;
      }
      if (hasButton && packet[1] == 1) {
        hasButton = false;
      }
    }
  }
  if (packet[0] == 2) {
    // IMU/HTC Vive control. Check the state of the buttons and change
the state
    if (packet[1] == 0 || packet[2] == 0) {
      state = 2;
      if (!hasSet) {
        // Need to make a copy of these values so we can set them back
        oldLx = Lx;
        oldLy = Ly;
        oldLz = Lz;
        oldOw = Ow;
        oldOx = Ox;
        oldOy = Oy;
        oldOz = Oz;
        hasSet = true;
      }
    } else {
      state = 0;
      hasSet = false;
    }
  }
}
```

```
// INVERSE KINEMATICS
void IK() {
  // These calculations follow the specifications set out in the
dissertation
  float d1 = s5 + t;
  t1 = atan((Ly - d1 * Oy) / (Lx - d1 * Ox));
  Quaternion q1;
  q1.w = cos(t1 / 2);
  q1.x = 0;
  q1.y = 0;
  q1.z = -sin(t1 / 2);
  Quaternion q1inv;
  q1inv.w = q1.w;
  q1inv.x = 0;
  q1inv.y = 0;
  q1inv.z = -q1.z;
  Quaternion L;
  L.w = 0;
  L.x = Lx;
  L.y = Ly;
  L.z = Lz;
  Quaternion Ldash = QMultiply(q1, L);
  Ldash = QMultiply(Ldash, q1inv);
  Quaternion O;
  O.w = 0;
  O.x = Ox;
  O.y = Oy;
  O.z = Oz;
  Quaternion Odash = QMultiply(q1, O);
  Odash = QMultiply(Odash, q1inv);
  float h1 = Ldash.z - d1 * Odash.z;
  float horiz = h - h1;
  float vert = Ldash.x - s1 - d1 * Odash.x;
  float d2 = sqrt(horiz * horiz + vert * vert);
  float d3 = s3 + s4;
  float alpha = acos((h - h1) / d2);
  float d2squ = d2 * d2;
  float d3squ = d3 * d3;
  float s2squ = s2 * s2;
  t2 = alpha + acos((d3squ - s2squ - d2squ) / -(2 * s2 * d2)) - PI / 2;
  t3 = acos((d2squ - s2squ - d3squ) / -(2 * s2 * d3));
  Quaternion qT1, qT2, qT3, qT13, qT13i, d3dash;
  qT1.w = cos(t1 / 2);
  qT1.x = 0;
  qT1.y = 0;
  qT1.z = -sin(t1 / 2);
  qT2.w = cos(t2 / 2);
  qT2.x = 0;
  qT2.y = sin(t2 / 2);
  qT2.z = 0;
  qT3.w = cos(t3 / 2);
  qT3.x = 0;
```

```
  qT3.y = sin(t3 / 2);
  qT3.z = 0;
  qT13 = QMultiply(qT1, qT2);
  qT13 = QMultiply(qT13, qT3);
  qT13i.w = qT13.w;
  qT13i.x = -qT13.x;
  qT13i.y = -qT13.y;
  qT13i.z = -qT13.z;
  d3dash = QMultiply(qT13, 0);
  d3dash = QMultiply(d3dash, qT13i);
  if (d3dash.y == 0.0) {
    t4 = 0;
  } else {
    t4 = -atan(d3dash.z / d3dash.y);
  }
  t5 = PI / 2 - (acos(d3dash.x / sqrt(d3dash.x * d3dash.x + d3dash.y *
d3dash.y + d3dash.z * d3dash.z)) - PI / 2);
  if(0x != 0.0){
    t6 = acos(0w) - t4;
  }else{
    t6 = -t4;
  }
}

// CONVERT THE IK ANGLES INTO A VALUE USABLE BY THE SERVOS
void ConvertAngles() {
  // Calculate the servo values
  servoValues[0] = (int)(((t1 + PI / 2) / PI) * 2200 + 450);
  servoValues[1] = (int)(((t2 + PI / 9) / PI) * 2200 + 450);
  servoValues[2] = (int)(((-t3 + PI) / PI) * 2200 + 450);
  servoValues[3] = (int)(((t4 + PI / 2) / PI) * 1600 + 700);
  servoValues[4] = (int)(((t5 + PI / 2) / PI) * 1600 + 700);
  servoValues[5] = (int)((t6 / PI) * 1600 + 700);

  // Limit the servo values
  for (int i = 0; i < 7; i++) {
    if (servoValues[i] > servoUpperLimits[i]) {
      servoValues[i] = servoUpperLimits[i];
    }
    if (servoValues[i] < servoLowerLimits[i]) {
      servoValues[i] = servoLowerLimits[i];
    }
  }
}

// QUATERNION MULTIPLICATION
Quaternion QMultiply(Quaternion q1, Quaternion q2) {
  Quaternion result;
  result.w = q1.w * q2.w - q1.x * q2.x - q1.y * q2.y - q1.z * q2.z;
  result.x = q1.w * q2.x + q1.x * q2.w + q1.y * q2.z - q1.z * q2.y;
  result.y = q1.w * q2.y - q1.x * q2.z + q1.y * q2.w + q1.z * q2.x;
  result.z = q1.w * q2.z + q1.x * q2.y - q1.y * q2.x + q1.z * q2.w;
```

```
  return result;
}

// CONVERT TO DEGREES
float ToDegrees(float angle) {
  return (angle * 180 / PI);
}
```

G.1.5 Unity code for HTC Vive (written in C#)

```
using UnityEngine;
using UnityEngine.UI;
using System;
using System.Collections;
using System.IO.Ports;
using System.Collections.Generic;
using Valve.VR;

public class enumports : MonoBehaviour
{
    // Set up variables to be linked in Unity
    public Button btn1 = null;
    public Button btn2 = null;
    public Button btn3 = null;
    public Button btn4 = null;
    public Text t = null;
    public Text d = null;
    public SteamVR_Input_Sources hand;

    public SteamVR_Action_Single trigger;
    //public SteamVR_Action_Vector2 trackPad = null;
    public SteamVR_Action_Boolean gripper;
    public SteamVR_Action_Pose pose;
    public SteamVR_Action_Vibration vibrate;
    public SteamVR_Action_Boolean up;
    public SteamVR_Action_Boolean down;
    public SteamVR_Action_Boolean left;
    public SteamVR_Action_Boolean right;

    // These variables do not appear in Unity
    private bool comSet = false;
    private bool ableToSend = false;

    private int ringTimer = 0;
    private SerialPort port = null;

    private byte[] packet = new byte[18];

    private bool trackStarted = false;
    private Vector3 startL;
```

```
    private Quaternion start0;

    // Use this for initialization
    void Start ()
    {
        // Enumerate the COM ports
        int i = 0;
        foreach (string s in SerialPort.GetPortNames()) {
            switch (i) {
            case 0:
                btn1.GetComponentInChildren<Text> ().text = s;
                break;
            case 1:
                btn2.GetComponentInChildren<Text> ().text = s;
                break;
            case 2:
                btn3.GetComponentInChildren<Text> ().text = s;
                break;
            case 3:
                btn4.GetComponentInChildren<Text> ().text = s;
                break;
            }
            i++;
        }

        // Set button listeners
        btn1.onClick.AddListener (SetCom1);
        btn2.onClick.AddListener (SetCom2);
        btn3.onClick.AddListener (SetCom3);
        btn4.onClick.AddListener (SetCom4);
        StartCoroutine (Delay ());
        packet [0] = 0x02;
    }

    // Update is called once per frame
    void Update ()
    {
        // Identify the Vive controller being used
        if (ringTimer < 5) {
            vibrate.Execute (0, 1, 320, 1, hand);
            ringTimer++;
        } else {
            ableToSend = true;
        }

        // Set up a COM port object
        if (comSet) {
            port = new SerialPort (t.text, 4800, Parity.None, 8, StopBit
s.One);
            port.Open ();
            comSet = false;
        }
```

```
        // Read the gripper and track the controller
        if (gripper.GetState (hand)) {
            packet [1] = 0x00;
            packet [2] = 0x00;
            TrackMovement ();
        } else {
            packet [1] = 0x01;
            packet [2] = 0x01;
            ZeroMovement ();
        }

        // Read the trigger
        packet [5] = (byte)(trigger.GetAxis (hand) * 255);
        ConvertTrackPad ();

        // Send data
        if (ableToSend && !comSet) {
            //d.text = packet [17].ToString ();
            port.Write (packet, 0, 18);
        }
    }

    // Call back functions for the COM port buttons
    void SetCom1 ()
    {
        t.text = btn1.GetComponentInChildren<Text> ().text;
        comSet = true;
    }

    void SetCom2 ()
    {
        t.text = btn2.GetComponentInChildren<Text> ().text;
        comSet = true;
    }

    void SetCom3(){
        t.text = btn3.GetComponentInChildren<Text> ().text;
        comSet = true;
    }

    void SetCom4(){
        t.text = btn4.GetComponentInChildren<Text> ().text;
        comSet = true;
    }

    // Motion tracker for Vive
    void TrackMovement ()
    {
        // Initialise tracking
        if (!trackStarted) {
            startL = pose.GetLocalPosition (hand);
```

```
            startO = pose.GetLocalRotation (hand);
            trackStarted = true;
        }

        // Extract the pose from the HTC Vive
        Vector3 currentL = pose.GetLocalPosition (hand) - startL;
        Quaternion q = pose.GetLocalRotation (hand);

        // Extract the orientation as angles
        Quaternion qinv;
        qinv.w = q.w;
        qinv.x = -q.x;
        qinv.y = -q.y;
        qinv.z = -q.z;
        Quaternion r = qinv * startO;

        // Prepare data for transmission
        int sOx = (int)(currentL.x * 30000);
        int sOz = (int)(currentL.y * 30000);
        int sOy = (int)(currentL.z * 30000);
        int rotx = (int)(r.z * 3600 / Mathf.PI);
        int roty = (int)(r.x * 3600 / Mathf.PI);
        int rotz = (int)(r.y * 3600 / Mathf.PI);
        //d.text = sOx.ToString () + " " + sOy.ToString() + " " + sOz.To
String();
        byte[] bytes = BitConverter.GetBytes (sOx);
        //d.text = sOx + " " + bytes[0].ToString () + " " + bytes[1].ToS
tring () + " " + bytes[2].ToString() + " " + bytes[3].ToString();

        // Construct packet
        packet [6] = bytes [1];
        packet [7] = bytes [0];
        bytes = BitConverter.GetBytes (sOy);
        packet [8] = bytes [1];
        packet [9] = bytes [0];
        bytes = BitConverter.GetBytes (sOz);
        packet [10] = bytes [1];
        packet [11] = bytes [0];
        bytes = BitConverter.GetBytes (rotx);
        packet [12] = bytes [1];
        packet [13] = bytes [0];
        bytes = BitConverter.GetBytes (roty);
        packet [14] = bytes [1];
        packet [15] = bytes [0];
        bytes = BitConverter.GetBytes (rotz);
        packet [16] = bytes [1];
        packet [17] = bytes [0];
    }

    // Set all tracking values to 0
    void ZeroMovement ()
    {
```

```
        for (int i = 6; i < 18; i++) {
            packet [i] = 0x00;
        }
        trackStarted = false;
    }

    // Convert the track pad to a packet
    void ConvertTrackPad ()
    {
        /*
        float x = trackPad.GetAxis (hand).x;
        float y = trackPad.GetAxis (hand).y;
        int leftTrack = (int)(y * 127) + 127;
        int rightTrack = (int)(y * 127) + 127;
        rightTrack = rightTrack - (int)(x * 127);
        leftTrack = leftTrack + (int)(x * 127);
        if (rightTrack > 255) {
            rightTrack = 255;
        }
        if (rightTrack < 0) {
            rightTrack = 0;
        }
        if (leftTrack > 255) {
            leftTrack = 255;
        }
        if (leftTrack < 0) {
            leftTrack = 0;
        }
        rightTrack = -rightTrack;
        leftTrack = -leftTrack;
        //d.text = leftTrack.ToString () + " " + rightTrack.ToString ();
        byte[] bytes = BitConverter.GetBytes(leftTrack);
        packet [3] = bytes [0];
        bytes = BitConverter.GetBytes (rightTrack);
        packet [4] = bytes [0];
        //d.text = packet [3].ToString () + " " + packet [4].ToString ()
;
        */

        // Get state of d-pad
        bool trackUp = up.GetState (hand);
        bool trackDown = down.GetState (hand);
        bool trackLeft = left.GetState (hand);
        bool trackRight = right.GetState (hand);

        // COnvert state to left and right values
        int leftTrack, rightTrack;
        if (trackUp) {
            if (trackLeft) {
                rightTrack = 0;
                leftTrack = 127;
            } else {
```

```
                if (trackRight) {
                    leftTrack = 0;
                    rightTrack = 127;
                } else {
                    leftTrack = 0;
                    rightTrack = 0;
                }
            }
        } else {
            if (trackDown) {
                if (trackLeft) {
                    leftTrack = 127;
                    rightTrack = 255;
                } else {
                    if (trackRight) {
                        rightTrack = 127;
                        leftTrack = 255;
                    } else {
                        rightTrack = 255;
                        leftTrack = 255;
                    }
                }
            } else {
                if (trackLeft) {
                    rightTrack = 0;
                    leftTrack = 255;
                } else {
                    if (trackRight) {
                        leftTrack = 0;
                        rightTrack = 255;
                    }else{
                        leftTrack = 127;
                        rightTrack = 127;
                    }
                }
            }
        }

        // Prepare and construct packet
        byte[] bytes = BitConverter.GetBytes(leftTrack);
        packet [3] = bytes [0];
        bytes = BitConverter.GetBytes (rightTrack);
        packet [4] = bytes [0];
        d.text = packet [3].ToString () + " " + packet [4].ToString ();
    }

    IEnumerator Delay(){
        yield return new WaitForSeconds (0.001f);
        if (port != null) {
            port.Write (packet, 0, 18);
        }
```

```
        }
}
```

Appendix H

H.1 Quaternion Mathematics

Quaternion mathematics is an extension of the complex numbers. Complex numbers are represented in the form $a + bi$ where i^2 is equal to -1. These numbers are often used to calculate rotations on the complex plane. The quaternion extension to the complex numbers includes 3 imaginary numbers i, j and k where $i^2 = j^2 = k^2 = -1$. Additionally, multiplication of 2 different imaginary numbers will result in the third the sign of which is determined by the order of the two numbers being multiplied as shown in Equation H.1:

$$\begin{aligned} i \times j &= k \\ j \times k &= i \\ k \times i &= j \\ j \times i &= -k \\ k \times j &= -i \\ i \times k &= -j \end{aligned} \tag{H.1}$$

From these imaginary numbers, it is possible to construct a quaternion of the form $w + xi + yj + zk$. To add two quaternions, add each element separately. To multiply two quaternions, apply the distributive law using the rules shown in Equation H.1

H.2 Properties of quaternions

One important property of quaternions is that they are not commutative as shown in Equation H.2 below:

$$q_1 q_2 \neq q_2 q_1 \tag{H.2}$$

Quaternions are distributive as shown in Equation H.3 below:

$$(q_1 + q_2)q_3 = q_1 q_3 + q_2 q_3 \tag{H.3}$$

Quaternions are associative as shown in Equation H.4 below:

$$(q_1 q_2)q_3 = q_1(q_2 q_3) \tag{H.4}$$

The length of a quaternion can be calculated as shown in Equation H.5 below:

$$l = \sqrt{w^2 + x^2 + y^2 + z^2} \tag{H.5}$$

A unit quaternion is a quaternion with a length of 1.

The conjugate of a unit quaternion is calculated by changing the sign of each element in the vector part. This is shown in Equation H.6 below:

$$q' = w - xi - yj - zk \tag{H.6}$$

The length of a robotic arm segment can be represented as a quaternion with a 0 value for w. This is shown in Equation H.7 below:

$$s_n = 0 + xi + yj + zk \tag{H.7}$$

A joint rotation can be represented by a unit quaternion constructed by the axis of rotation within the vector as it appears on a unit sphere. An angle of rotation, θ, is incorporated by setting w to $\cos(\theta/2)$ and multiplying each element in the vector part $\sin(\theta/2)$. This is shown in Equation H.8 below:

$$r_\theta = \cos\frac{\theta}{2} + xi\sin\frac{\theta}{2} + yj\sin\frac{\theta}{2} + zk\sin\frac{\theta}{2} \tag{H.8}$$

Quaternions are often represented as a rotation component (w) and a vector component as shown in Equation H.9 below

$$q = (w + \vec{v}) \tag{H.9}$$

To rotate a segment using a quaternion, represent the segment as the vector part of a rotation quaternion with $w = 0$ as shown in Equation H.7. The resulting rotation can be calculated as shown in Equation H.10 below:

$$\overrightarrow{v_2} = q\ \overrightarrow{v_1}\ q' \tag{H.10}$$

If there is more than one rotation to be applied to a segment, these quaternions can be multiplied together to form an equivalent rotation quaternion. This simplifies complex rotations by concatenating them into single rotations. This is shown in Equation H.11 below:

$$q_{12} = q_1\ q_2 \tag{H.11}$$

The conjugate for H.11 for rotation purposes preserves the non-commutative property through another property shown in Equation H.12 below:

$$q'_{12} = q_2{}' \, q_1{}' \tag{H.12}$$

Finally, orientations can be represented by the form shown in Equation H.11. Such orientations a with reference to the quaternion $q = 1$. This quaternion is known as the identity quaternion and as such any quaternion multiplied by it will result in itself. Thus the quaternion $q = 1$ represents no rotation.

Appendix I

I.1 Artificial Neural Networks

Artificial neural networks are modelled on real neural networks such as those found in the brains of animals. Brains are made from neurons. A neuron has several parts and the relevant parts are listed below:

1. Dendrites: Provide a conduction path from the synapses to the cell membrane
2. Cell membrane: Conveys the signal from the dendrites to the axon
3. Axon: provides a conduction path from the cell membrane to the synapses
4. Synapses: Allows a certain amount of conduction between two neurons

From these parts, an artificial neuron can be modelled. Dendrites are modelled as input values from previous neurons or from sensors etc. which are summed up. The cell membrane is modelled as a transfer function. Natural neurons have a transfer function that closely resembles the sigmoid function. The artificial neuron has an additional property called the bias which ensures certain functionality. The axon is modelled as a variable containing the result of the transfer function. This variable is passed on to the next neurons. The synapses are modelled as weight values which are multiplied by the input from the previous neurons.

Artificial neural networks are usually organised into layers. There is always an input layer followed by at least one “hidden” layer followed by an output layer. The number of neurons in each layer and the number of hidden layers depends on the processing requirements.

Figure 48 below shows an example of an artificial neural network containing a single hidden layer:

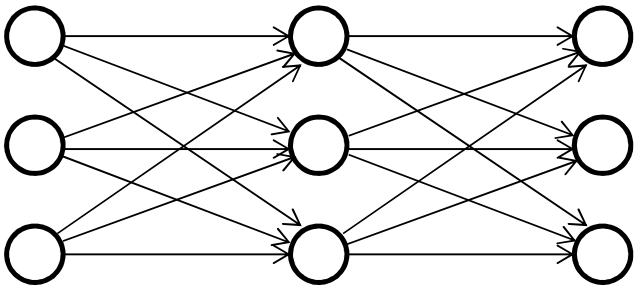


Figure 48 - Artificial neural network

The input to a neuron is shown in Equation I.1 below:

$$I = \sum_i w_i y_1 + b \tag{I.1}$$

Where w_i is the weight from an input, y_i is the input and b is the bias for the neuron.

The input is passed through a sigmoid routine shown as Equation i.2 below:

$$\sigma = \frac{e^x}{e^x + 1} \tag{I.2}$$

The output from an artificial neural network of this type is defined as a classification. That is the value at each output neuron represents a probability that the decision that neuron represents is the true representation. For example, each output neuron could represent different letters of the alphabet. In this situation, the input could be a set of pixels representing hand written letters.

Artificial neural networks are trained, not programmed. There are several options when it comes to training artificial neural networks, but the most common is the back propagation method. This method looks at the change in a neurons output given a change in weight. As such, the derivative of the sigmoid function is required. Fortunately, the derivative of the sigmoid function contains the sigmoid function which simplifies the back propagation method. The back propagation method starts at the outputs and adjusts the weights for the inputs to this layer. It then moves onto the hidden layer and adjusts the weights to this layer.

Finally, once the artificial neural network has been trained, it is operated in forward propagation mode where by input signals are passed through the neural network to provide an output.